CONVERSION OF COMPUTER PROGRAMS
TO
DECISION TABLES

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

BY

VIRENDRA GUPTA

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

NOVEMBER, 1969

Alas! My PARENTS are not alive today!!!

CERTIFICATE

This is to certify that this work on 'Conversion of Computer Programs to Decision Tables' has been carried out under my supervision and it has not been submitted elsewhere for a degree.

V. Rajaraman
Professor of Electrical Engineering &
Head,
Computer Centre
Indian Institute of Technology
KANPUR

ACKNOWLEDGEMENTS

The author would like to express his deep sense of gratitude to Professor V. Rajaraman who served as thesis supervisor. He has been and continues to be a source of inspiration. He found time and patience for many valuable discussions and has been a constant source of encouragement.

The author is grateful to Dr. B. Prasada, Head, Electrical Engineering Department for providing necessary facilities, to the entire staff of the Computer Centre, IIT-Kanpur for their cooperation, helpful discussions and advise; to Drs. M.K. Verma (Department of Humanities, IIT-Kanpur, presently with Department of Linguistics/Indics, University of Wisconsin) and S.V. Rangaswami (Computer Group, Tata Institute of Fundamental Research, Bombay) for some very stimulating discussions, and to all his friends, in particular his 'Guru Bandhus' Dr. C.R. Muthukrishnan, and Shri T. Radhakrishnan for showing a keen interest, understanding and help in many ways. Shri A.C. Agarwal has done a good job of typing this thesis.

Finally, the author would like to express his sincere heartfelt thanks to his life companion, Usha, who for 2 years had to share her husband with a machine. But for her ocean like patience, cooperation, and constant encouragement, this venture would not have been possible.

SYNOPSIS

Of the

Ph.D. Dissertation

on

CONVERSION OF COMPUTER PROGRAMS

TO

DECISION TABLES

by

VIRENDRA GUPTA
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

November, 1969

Two important, seemingly different problems facing computer scientists have been the detection of logical errors in programs and the problem of converting programs written for one computer to to that of another one. This thesis proposes the use of decision tables as an intermediate step in solving both these problems.

If algorithms for conversion of computer programs to decision tables were available then the resulting decision tables would be very useful for detecting errors in logic. As programs for converting decision tables to machine language are available, decision tables would be a good intermediate language in converting programs written for one computer to those for another. This thesis attempts to develop such algorithms.

Most of the literature on decision tables deals with conversion of computer programs to decision tables. Applications of decision tables to file handling and system design among others have also been discussed in literature.

The approach in this thesis has been to use decision tables:

(i) as an intermediate language in conversion of programs written for one computer to that of another, and

(ii) as an aid in debugging and documentation.

These are the areas in which the potentialities of decision tables have not been explored so far. The following are the main results reported in this thesis.

(i) An algorithm to obtain decision tables corresponding to syntactically correct FORTRAN programs has been developed.

The given program is scanned from the first statement. While scanning, two tables A and B are formed. All the control statements are entered in Table A and all the executable statements, with statement numbers in table B. Using the above two tables, the algorithm gives a systematic procedure for obtaining the decision table.

Based on the above algorithm, LOGITRAN, a translator to convert FORTRAN programs to decision tables has been developed. This translator takes syntactically correct FORTRAN programs as input and produces the corresponding decision table as output.

LOGITRAN has been written in FORTRAN-IV and has a high degree of machine independence. If a large decision table is fed as input to LOGITRAN, it would also parse it into smaller tables.

(ii) The applications of Boolean matrices associated with a flowchart or a computer program are critically analysed and their shortcomings discussed. A Structure matrix is proposed as a better and more general representation of program structures.

(iii) Based on the above mentioned structure matrix, an alternative procedure to get the decision tables has been developed.

(iv) MATDT, a scheme, which combines the potentialities of matrices and decision tables is proposed as an effective debugging and documentation aid.

(v) A method is suggested whereby some of the difficulties of translation of assembly language programs can be bypassed by using decision tables as an intermediate language.

An algorithm to convert assembly language programs to decision tables is developed. The decision table alongwith a certain amount of pre and post editing, it is felt, would lead to some useful ideas for solving the decompiler problem.

CONTENTS

CHAPTER I

INTRODUCTION

Two important, seemingly different problems facing computer scientists have been the detection of logical errors in programs and the problem of converting programs written for one computer to that of another one. This thesis proposes the use of decision tables as an intermediate step in solving both those problems.

If algorithms for the conversion of computer programs to decision tables were available then the resulting decision tables would be very useful for detecting errors in logic. As programs for converting decision tables to machine language are available, decision tables would be a good intermediate language in converting programs written for one computer to those of another. This thesis attempts to develop such algorithms.

Given a computer program, the problem of analysing it has long existed. The purpose of analysis may be to understand a program produced by another person, to document a program, or to debug one. It is worthwhile to automate this process of debugging and documentation, that is to develop computer programs which would analyse a user's program and display logic implemented by the program.

The obsolescence time of computers have been about 5 years and this has created the problem of running programs written for an old computer on a new one. Economy of investment

in terms of time and money requires that this be done with a minimum extra effort.

1.1. History of the Problem:

For getting the program logic, Flowcharts have been traditionally used. A number of techniques have been discussed in the literature (2,22,25,27,32,51,63,80,84,85) for generating flowcharts with computers. Flowcharts, particularly computer generated ones, have a number of deficiencies which make them poor vehicles of communication of program logic. In complex problems a clear and obvious flow of logic from START to STOP is difficult to detect with flow charts. This difficulty is compounded in computer generated flowcharts where the limitation of available character sets and printer space make the charts extend over a number of pages and make them unreadable. Further if an analyst wants to find out whether all conditions have been accounted for or if alternate superior methods exist, flowcharts are of little use. Another difficulty has been that flowcharts can not be used as a direct input to a machine (90).

Decision Tables (D.T.s) overcome some of these difficulties (1,6,8,12,45,56,60). These tables define clearly and concisely the program logic in a tabular form and separate the conditions tested from the actions executed. The tabular structure facilitates debugging of program logic. Further the printer output

can be presented in a format which is eminently readable.

It would be worthwhile to automate the conversion of programs to decision tables. When such a translator is available, then programmers may use it as a debugging aid to detect errors in program logic or to have an up-to-date documentation of the program. For assembly language programs the decision tables obtained may be used as an intermediate language in conversion besides their use for documentation and debugging.

1.2 Program Debugging:

1.21 Run Time:

There are two types of errors which can occur in computer programs. They are:

i) Syntax errors

ii) Semantic errors

Syntax errors are easily detected during compilation by the translator and error messages given to the user. Semantic errors or errors in logic, however, are more difficult to detect. In most programs at least a third of the total time is spent in detecting errors in logic. The problems involved in debugging programs may be approached at several different levels and in several different ways. In the case of a compiler program, they may be approached either at the level of the absolute object code as loaded into machine, or at the level of some intermediate language or at the level of compiler source language. They may be

approached by extracting information from the computer console, providing check point information for comparison with intermediate results (18,40) or may be approached by post mortem or snap dumps (39) or by tracing sections of the program (15,37). Data too, may be erroneous and have to be taken care of (44).

Some of the standard diagnoastic procedures were described above. These procedures require that the programmer real in sample data with his program, and then follow it up through the program by means of memory dumps, break point printing and similar procedures. It must be noted however that these procedures are dependent upon the test data, a proper choice of which in itself is quite an arduous task. Senko (81), Miller and Maloney (58) have described in detail about the selection of test data and checking of subroutines. The presently available diagnoastic procedures are usually time consuming. They are confined to the path of flow taken up under the specific conditions existing at the time of the particular run. Thus, a part of the program may never be encountered and a typical path of flow may not be seen.

1.22 Debugging Logical Errors:

Information can also be given about a program without running it. One such technique, namely, Flowcharting has been mentioned before. Techniques based on Boolean Matrices, associated with program flowcharts, have been discussed by Prosser (71) and Merimont (54). Elementary manipulations on

these matrices are shown to yield detailed information concerning the internal logical consistency of the flow diagrams. A number of inconsistencies like redundant statements, and endless loops can be brought out by manipulating these matrices. Decision Tables, generated from computer programs by a translator program is a new debugging technique proposed in this thesis. Decision tables because of their tabular structure help in pinpointing ambiguities and contradictions in the statement of the logic of the problem.

Starting with a syntactically correct program, getting and analysing its structure with matrices, and finally converting it to a decision table is proposed as a very effective aid for debugging logical errors in programs, without actually 'running' them.

1.3 Overview of the Literature:

The purpose of this section is to review the literature that has appeared in the area of (i) program analysis and (ii) conversion of programs written in lower level language to a higher level language. The review is intended to place the method presented in this thesis in a proper perspective.

1.31 Literature on Program Analysis:

Karp (41) has shown that the properties of directed graphs and the associated matrices can be used to detect errors

and eliminate redundancies in programs. His paper was the first one proposing graph theoretic models for computer programs. He has also shown how these properties can be used in the synthesis of composite diagrams.

His procedure requires enumerating all the proper paths and thus it is felt that any automation of the procedures suggested by him would be wasteful of time.

Prosser (71) describes the analysis of directed graphs corresponding to programs or flowcharts, by the use of Boolean matrices. Two such matrices are associated with each graph; the first is called the connectivity matrix, containing the topological structure; the second is called the precedence matrix and contains the precedence relations. Prosser's technique consists of raising the matrices to higher powers until some power of the matrix is a null matrix. By adopting his procedure one can get an idea about program segments without exits, segments in loops and also if the program can be decomposed into relatively independent sub-programs.

Although, we have to deal with Boolean matrices in Prosser's technique, yet it must be realised that raising them to higher powers is a time consuming job.

Warshall (91) has proposed an algorithm for manipulating Boolean matrices, where running time goes slightly faster than square of n (nXn is the size of the matrix) as compared to cube of n in the normal procedures.

Merimont (54) has shown how the presence of a contradiction can be revealed, without making explicit any of the implications of the original set, thus eliminating much tedious computation. He describes the physical significance of raising Boolean matrices to higher powers. His criterion for consistency is stated as follows "A set of precedence relations is consistent, if and only if, every principal submatrix has at least one zero row or zero column".

Based on this principle he gives a very simple procedure for the search of contradictions in the program. A more detailed discussion of this procedure will be presented in chapter IV where the conversion of programs to decision tables via matrices is described.

Meakawa's (53) article on Automatic Flowcharting needs a special reference. Although this paper is difficult to read and the flowcharts obtained are not elegant, it is the only paper on automatic flowcharting with an analytic approach and attention to semantics. His is the only significant paper in the open literature on the flowcharting of assembly languages. In an earlier attempt flowcharts for assembly language were discussed by Knuth (51), and his technique requires the user to provide comments (Remarks field) in a certain definite format, which is too stringent a requirement.

Meakawa, after describing the principles on the basis of

which several instructions can be combined, deals with program flow description using a list-notation. The notation used is an extension of the one introduced by Krider (50). Meakawa's technique helps in combining smaller blocks into bigger blocks, and the levels of flowcharting are exhaustive.

Some of the ideas of Meakawa and Krider will be discussed again in the last chapter where it is shown how they can be used in program structure analysis and simplification.

1.32 Literature: Translation of Programs Written in Lower Level Languag

Techniques like simulation (17) and emulation which aid in conversion of programs, written for one machine to be run on other machines, have been dealt in great detail by Benjamin (5), and Tucker (86). How the problem of conversion of programs has influenced the hardware of modern machines has been discussed by Mc Cormack et.al. (55).

In the present context of conversion of programs written in lower level languages to higher level languages, it is worthwhile mentioning that the difficulties involved in the former are very much similar to those encountered in the machine translation of natural languages.

Bar-Hillel (4), and Koulagina (48) have described in detail the various aspects of the problem of machine translation of natural languages. Analysis of structure and the interpretation of meanings are the two main problems in M.T. The problem of structure,

that is syntax has been studied in great detail in recent years. However, the research in semantics is still in an infant state (67). From the present state of the art in M.T., it is felt that the most difficult problem being faced is the multiple meaning of words and phrases.* This is intrinsically the problem of semantic theory (42). Nagao (62) has suggested that syntax and semantics, although they have been studied separately, demand a unified treatment if the intrinsic natures behind the sentences is to be clarified.

Coming to problems of translation of assembly language programs, Gains (19) and Halpern (27,28) have discussed the same in some detail, although from two different angles. They have identified the following problems of translation of computer languages:

i) Untranslatability

ii) Idiomatic expressions

iii) Program self modification

iv) Differences in structure of the source and the target machines.

Because of the above mentioned difficulties, Gains has proposed the machine translation of a programmer pre-edited source deck, followed by programmer post-editing or debugging to be a reasonable goal for automatic (to be more precise, it should be called 'semi automatic') translation. Thus, what seems to be a realistic solution of this problem is one in which majority of the work is in the middle step, with programmer intervention at either

---

* Appendix I gives a few examples explaining this point.

end being held to a minimum. Gains does not mention of anything regarding implementing his ideas. He seems to be too pessimistic.

A few reports of the translation efforts that went beyond simulation of the source machine, or use of macros in translation (11) have appeared. Gunn's (24) is a good illustration of how far one can go simply by proceeding in a fairly simple manner: simulation of certain features of the source machine and instruction by instruction translation of the other statements of the source program.

Gunn did not consider problems of untranslatability or idiomatic expressions at all. He lumped the two as programmers tricks .

Opler's et.al . (65) is an effort of significant importance. They have dealt in detail how they converted programs written for IBM 705 for IBM 7074. They have described in detail how dynamic execution of the source program provided a trace information about its statements and instruction modification.

The authors have concluded, that not all IBM 705 programs could be translated. They did not care for semantics and never made any attempt to determine the actual meaning in terms of the original problem and did not make any attempt to follow the hardware logic of the source machine translation.

Sassaman (77) describes a computer program which translates assembly language decks for the IBM 1700 series into FORTRAN. The author does not describe any algorithm which aided in the translation process.

Oslen's (66) paper on translation of programs from Philco 2000 series for IBM 7094 at problem oriented, symbolic language and binary levels seems to be one of the best in this area. He has given some statistics about the number of statements of Philco programs, which had to be converted manually, because of some of the difficulties mentioned before (19).

1.4 <u>New Results</u>:

Most of the literature on Decision Tables deals with conversion of computer programs to decision tables (3,14,20,21,34, 45,46,47,60,61,67,68,73,74,88,89). Applications of Decision Tables (9,12,33) to file handling (13,16,23), information retrieval (52), system design (6), inventory control (79), process control (43,64), power system analysis (50), compiler writing (73), machine design (31) and switching theory (82) have been discussed in literature.

The approach in this thesis has been to use decision tables as an <u>intermediate language</u> in conversion and as an aid in <u>debugging</u> and <u>documentation</u>, an area in which the potentiality of D.T. s has not been explored so far.

The following are the main results reported and represent the original contribution in this thesis:

1. An algorithm to obtain Decision Tables corresponding to syntactically correct FORTRAN programs has been developed.

2. An alternative Boolean procedure, based on the <u>structure</u> matrix has been developed.

3. An algorithm to obtain Decision Tables for assembly language programs has been developed.

4. An effective debugging scheme MATDET utilising the properties of Boolean Matrices and Decision Tables is proposed.

1.5 <u>Outline of the Thesis:</u>

The thesis consists of 5 chapters. Chapter II describes the algorithm for converting FORTRAN programs to Decision Tables.

Chapter III describes the implementation of the above mentioned algorithm. A number of examples are included which bring out how decision tables serve as excellent tools for debugging and documentation.

In chapter IV after discussing the short comings of Precedence and Connectivity matrices, <u>Structure matrix</u> is proposed as a better and more general form of representation of program structures and an algorithm to arrive at a decision table from this structure matrix is presented.

In chapter V a method is suggested whereby some of the difficulties of translation of assembly language programs (section 1.32) can be by passed by using Decision Tables as an intermediate language.

CHAPTER II

AN ALGORITHM FOR CONVERTING FORTRAN PROGRAMS

TO

DECISION TABLES

2.1 Introduction:

Program analysis reveals two distinct aspects

i) computational

ii) logical or decision making

Numerical analysis and related areas are concerned with computation and little decision making. In business data processing the emphasis is on procedures invoking complex decision logic. Problems of this nature are frequently encountered in file handling, compilers, string processors, process control, inventory control, information retrieval and machine design. Since the logical development of the problem is affected only when the program is required to select between two or more possible paths, it is only the decision elements that are of importance to us in our present context. For this reason it is advantageous to consider representations of the problem-flow that omit the descriptions of the numerical steps. The logical tree is the simplest form of representation from this point of view. The algorithm explained in the next section is just a systematic procedure for traversing the various paths of the tree from start to terminals.

2.2. Algorithm:

An algorithm (26) which reduces a syntactically correct FORTRAN program to decision table, is explained here with the aid of an example.

Example: A student decides to go on a picnic on Sunday if

C	STUDENT - HOMEWORK/PICNIC PROGRAM	S.No.
	READ 10, HWORK, WETHR, PENDNG, CLEAR	1
	IF (HWORK - PENDNG) 20, 30, 20	2
20	IF (WETHR - CLEAR) 40, 50, 40	3
50	PRINT 60	4
	STOP	5
40	PRINT 70	6
	STOP	7
30	PRINT 80	8
	GO TO 20	9
60	FORMAT (12HGO ON PICNIC)	10
70	FORMAT (12HGO TO TEMPLE)	11
80	FORMAT (25HCOMPLETE HOMEWORK MORNING)	12
10	FORMAT (4A6)	13
	END	14

Fig. 2.1 FORTRAN-II PROGRAM. Student Homework Picnic Problem

no homework is pending and the weather is clear. If weather is not

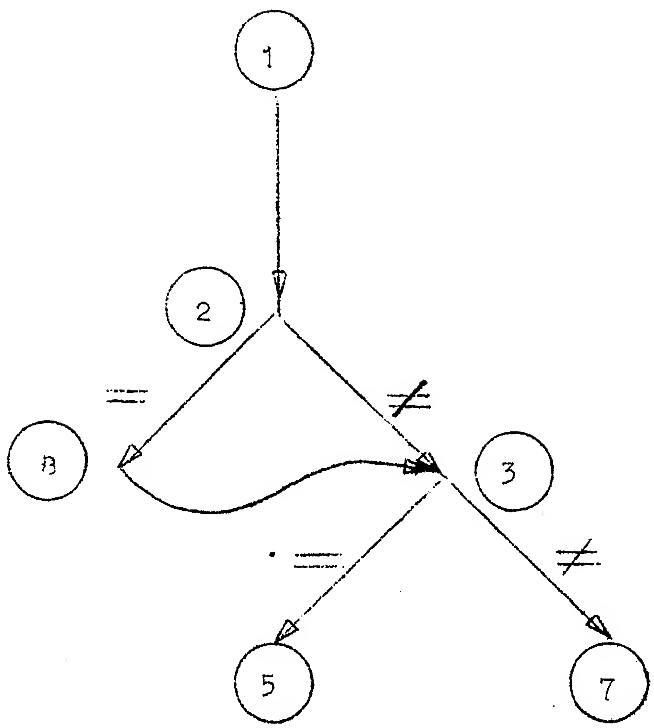


Fig. 2.2 Logical Tree - Student Homework/Picnic Problem

clear and no homework is pending, he decides to visit a temple. If the weather is clear and homework is pending, he resolves to complete the homework in the morning and go on a picnic in the afternoon.

A FORTRAN-II program to implement the above word statement is given in Fig. 2.1. The logical tree for this program is given in Fig. 2.2. The algorithm could be understood by considering the flowchart corresponding to the program given in Fig. 2.3.

Starting with the first condition the appropriate path is traced till the STOP statement is reached. On its way to STOP, at each condition state (decision element), the internal statement number which lead to the current state is recorded. Thus the path may be retraced from STOP, back to the previous node, appropriate path for the next STOP traced, and the procedure repeated. This way what we are doing is tracing the different paths of the logical tree.

The Algorithm:

The given program is scanned from the first statement. While scanning, two tables A and B are formed. All the control statements except DOs are entered in table A and all executable statements with statement numbers in table B. As DO loops by themselves do not affect the logic of the program, these are not considered in table A.

Reference:-

ISN - S.No.
SN - STMNT

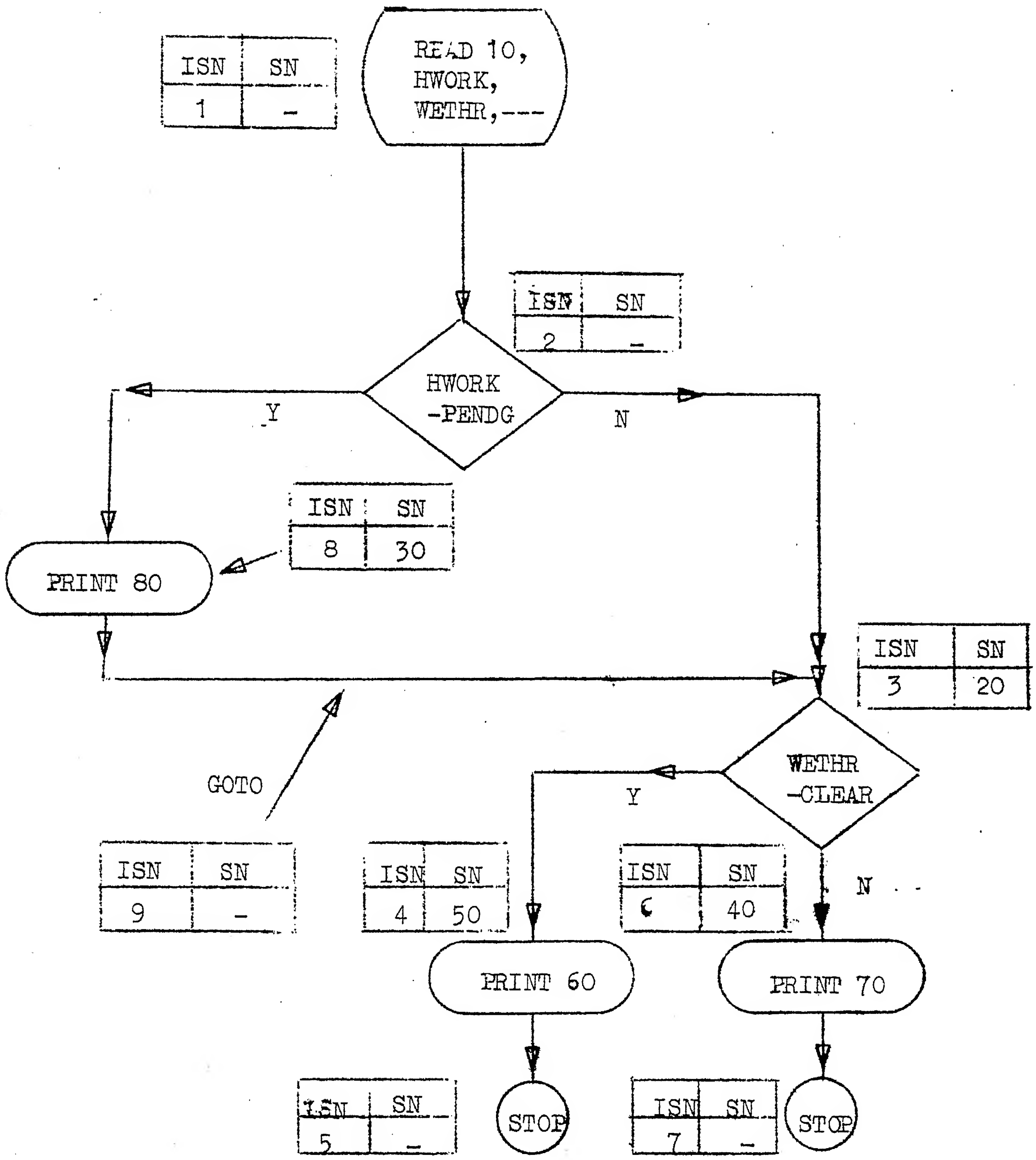


Fig. 2.3 FLOW CHART - Student Homework/Picnic Problem.

Table A consists of 7 columns. The first column contains the nature of the control statement. For IF statement, the expression within the paranthesis is entered. GOTO and STOP/RETURN are entered as they appear. Column 2 has the serial number of the statement. Column 3 has the STATEMENT number of the statement if it has one, otherwise it is a blank. In columns 4,5 and 6, the three branches of the IF statement are entered. As GOTO is an unconditional transfer to the same statement, this statement number is entered in all the three columns (4,5 and 6). As STOP has no "transfer to" statement number, columns 4,5 and 6 are blanks. The seventh column is used as temporary storage. Table A corresponding to the example program given before is shown in Fig. 2.4.

The second table (Table B) has four columns. In the first column the executable statement as it appears in the program is entered. The second and third columns are similar to the corresponding columns of Table A. The fourth column is used as temporary storage. Table B corresponding to the sample program is shown on the next page. (Fig. 2.5).

TABLE A

Column No.

1	2	3	4	5	6	7
Condition	S.No.	Statement No.	Br 1	Br 2	Br 3	Temp
HWORK - PENDNG	2		20	30	20	
WETHR - CLEAR	3	20	40	50	40	2 2
STOP	5	–	–	–	–	3 3
STOP	7		–	–	–	3 3
GOTO	9		20	20	20	2

Fig. 2.4 TABLE A for Program Fig. 2.1

TABLE B

1	2	3	4
Statement	S.No.	Statement No.	Temp
PRINT 60	4	50	3 3
PRINT 70	6	40	3 3
PRINT 80	8	30	2

Fig. 2.5 TABLE B for Program Fig. 2.1

Using the above two tables, the decision table is devel o ped as follows:

1. Scan the first row of Table A. Enter condition in the condition stub of decision table. See if statement number in Br 1 is equal to either Br 2 or Br 3. If yes, circle the appropriate entry. Enter .NE.0 in condition entry if Br1 = Br3. If Br1 = Br2 enter .LE.0 In all other cases it will be .LT.0.

2. Match the statement number of Br 1 against entries in column 3 of Table A. Go to the matching row. Enter in column 7 of matching row the serial number of the statement from which the present statement was reached. In this example row 2 matches. As we came from serial No.2 a'2' is entered in TEMP (Table A). Enter in condition stub of decision table the contents of column 1, unless it is for a STOP or GOTO; In this case row serial No.2.

3. Scan Br 1, Br 2, Br 3 (of matching row) for equality, circle equal ones. Repeat step 2. In this case no statement number in Table A matches with 40. In such a case go to Table B and match with column 3 of this table. Enter in Temp of Table B the serial number of statement in Table A which led to it. Enter matching entry statement number as an action in the decision table.

4. See the serial number of this statement. Go to the next higher serial numbered statement in Table A. Enter in TEMP the serial number of statement in Table A which led to it. (In this case 3). If the statement is STOP, enter it as an action entry. If it is a condition, enter the condition and proceed as in step 2.

In case of STOP, the next rule is to be found. For this go to the row whose serial number is in column 7 (Viz. 3 in this case).

5. Take the next uncircled branch of this row (Br 2 of row serial number 3 of Table A). Scan Table A, column 3 (statement number) for a match for this branch. If no match found, go to Table B for a match. Repeat step 4. In this case serial number 4 in Table B matches. It leads to STOP. The Temp. entry is 3 and it leads to serial number 3. As all branches have been considered (all branches of this row circled), one goes to the row serial number 2 as given in TEMP., after erasing the circle around Br 1, Br 2, Br 3 of the most recently tested condition.

6. The uncircled branch of the condition taken up is 30. It is circled and the branch taken. It leads to Table B serial number 8, which leads to 9 in Table A. The branches of 9 are all equal (GOTO). It is thus an unconditional branch and leads to serial number 3. The branches of this statement are taken up one at a time and the same steps as before are repeated.

7. When all branches are scanned, control is returned to serial number 2 which is the first condition statement in the program. As all branches of this statement have been exhausted the procedure is complete.

The resulting DECISION TABLE for this sample program is shown on the next page.

DECISION TABLE

ENTRY	RULE			
Condition	1	2	3	4
HWORK - PENDNG	.NE.0	.NE.0	.EQ.	.EQ.0
WETHR - CLEAR	.NE.0	.EQ.0	.NE.0	.EQ.0
Action				
GOTO	40	50	30	30
GOTO	STOP	STOP	40	50
GOTO			STOP	STOP

2.21 DO Loops:

It has been mentioned before that DO statements are not entered in Table A: our aim in program analysis is to get the program logic. For this our approach is to consider conditions such that every path will be traversed at least once and, furthermore, that it will be entered from every possible entry point. If these conditions are satisfied, we shall know that the work performed along a path is correct and that the conditions at each of its entry points are the proper ones for sucessful completion of the path. Under this premise it would be unnecessary to go through each loop more than once, unless dictated by an entry, from another path. A single traverse through the loop will give us the logic. This means that if we

have specified conditions to cause the program to exit once from each side of a branch point, it will be inconsequential to us, in so far as the program logic is concerned, whether the program subsequently loops back to a previous point in the program or goes to a stop. In the tree and the table, a path that leads into a loop will appear the same as one that leads directly to a stop, in so far as the debugging requirements are concerned.

In numerical analysis and iterative type problems, some steps of computation amount to the object time modification of the condition tested within a DO loop. Such cases are discussed in section 2.4.

Action Set:

The actions are written in the order in which they are executed.

Above is a simplified version of the procedure used for converting FORTRAN programs to DECISION TABLES. However, statements like computed and assigned GOTO have not been dealt with in the above description. Many such problems are encountered in FORTRAN programs and are dealt in the next section.

2.31 Computed and Assigned GOTO:

These are taken as a sub-decision table and linked to the main table. For example,

```
10  GOTO  (20,30,40,25,40),I
```

can be expressed as shown on the next page.

SUBTABLE 10

I	1	2	3	4	5
GOTO	20	30	40	25	40

Similarly

16 GOTO INDEX, (20,40,43)

can be expressed as,

SUBTABLE 16

INDEX	20	40	43
GOTO	20	40	43

2.32 Extensions for FORTRAN-IV:

Logical IFs :

In the case of logical IFs only one branch is explicitly specified if it is a GOTO some statement number. The other branch is implied as the next serial statement. This can be taken care of by using serial numbers of various statements for finding a match in step 2, Section 2.2.

Complex Logical IF

Complex logical IF s can themselves be considered as a decision table and are treated similarly. The sub-decision table

is entered in table A and properly linked to the remaining DECISION TABLES. For example

```
11      IF(A.EQ.B. AND.C.LE.D.OR.A.NE.D.AND.F) GOTO 25
```

will give rise to

SUBTABLE 11

	1	2	ELSE
A - B	.EQ.O		
C - D	.LE.O		
A - D		.NE.O	
F		T	
GOTO	25	25	Next serial number

In the above decision table, the relational operators. are picked from the program statement e.g. $\leqslant$ zero $\equiv$ .LT.O etc. etc. and for logical variables a T is entered for .TRUE.

2.4 Non-Tree Structured Programs:

The algorithm given in the last section works well for tree-structured programs like compilers and those encountered in business data processing problems. However, the previous algorithm fails in iterative computational problems, such as those encountered in numerical analysis. This is due to the fact that the algorithm does not distinguish between paths due to loops and other paths, and

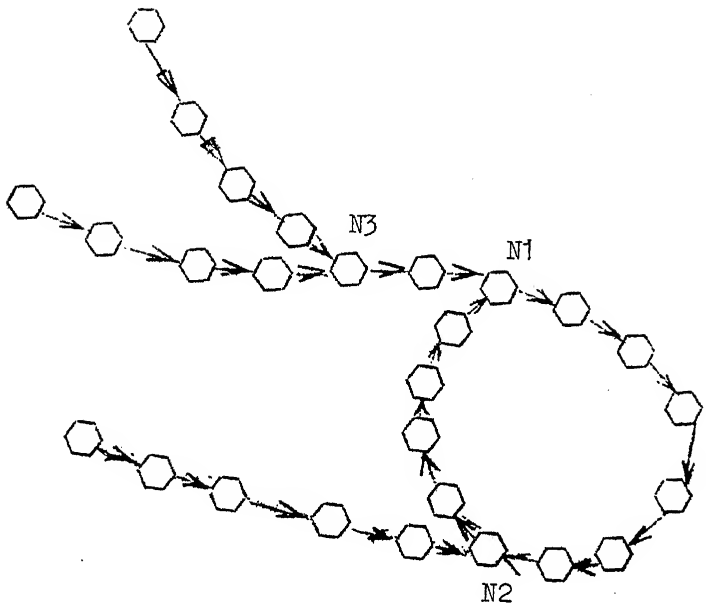


Fig. 2.6 Difficulties: Non-tree Structured Program.

```
C       SAMPLE PROGRAM GAUSS SEIDEL
        DIMENSION A(3,4),B(3,4),X(3,22)
12      PRINT 41
        READ 10,N,MAX,CLOSE,K,M,NO
10      FORMAT(2I5,F10.3,3I5)
        READ      ,(X(I,1),I=1,N),((A(I,J),J=1,M),I=1,N)
        DO 11 I=1,N
        DO 11 J=1,M
11      B( I,J)=A(I,J)/A(I,I)
C       KEPT FOR THE SAKE OF TRIAL
    22  I=1
133     J=1
        X(I,K+1)=B(I,M)
C25     IF(I-J) 17,16,17
25      IF(I-J) 17,16,57
C       KEPT FOR THE SAKE OF TRIAL TO REDUCE THE SIZE OF DECISION
C       TABLE
15      X(I,K+1)=X(I,K+1)-B(I,J)*X(J,K+1)
        GOTO 16
17      X(I,K+1)=X(I,K+1)-B(I,J)*X(      J,K)
16      IF(J-N)  26,27,27
26      J=J+1
        GOTO 25
27       IF(I-N)  37,38,38
37      I=I+1
        GOTO 133
38      L=K+1
        DO 21 I=1,N
        ABSFX=ABS (X(I,K+1)-X(I,K)   )
        IF(ABSFX-CLOSE)  21,21,47
47      IF(K-MAX)  56,57,57
21      CONTINUE
        PRINT  991,(X(I,L),I=1,N),K
        NO=NO+1
        IF(NO-2)  12,12,97
56      K=K+1
        GOTO 122
57      PRINT 43
41      FORMAT(/1H ,*--------- ---------------*)
43      FORMAT(/1H ,* PROGRAM FAILS TO CONVERGE*)
991      FORMAT(1H ,5X4HX1 =F12.5,5X4HX2 =F12.5,5X4HX3 =F12.5,
       10H    K=I3)
97      PRINT 41
99      STOP
        END
```

FIG.2.7 <u>SAMPLE PROGRAM-GAUSS SEIDAL METHOD</u>

DECISION TABLE

RULES

TION	1	2	3	4	5	6	7	8	9	10	11	12	13
J	.LT.0	.LT.0	.LT.0	.LT.0	.LT.0	.LT.0	.EQ.0	.EQ.0	.EQ.0	.EQ.C	.EQ.P	.EQ.0	.GT.0
N	.LT.0	.GE.0	.GE.0	.GE.0	.GE.C	.GE.0	.LT.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	
N		.LT.0	.GE.0	.GE.C	.GE.0	.GE.0		.LT.0	.GE.0	.GE.0	.GE.0	.GE.0	
X)) E)			.LE.0	.LE.0	.GT.0	.GT.C			.LE.0	.LE.0	.GT.0	.GT.0	
2			.LE.0	.GT.0					.LE.0	.GT.0			
AX					.LT.0	.GE.0					.LT.0	.GE.0	
NS													
	17	37	38	97	56	57	26	37	38	97	56	57	57
	26	133	21	STOP	122	STOP	SELF	133	21	STOP	122	STOP	STOP
	SELF	SELF	12		SELF			SELF	12		SELF		
			SELF						SELF				

Fig. 2.8 Decision Table for Program of Fig. 2.7.

goes astray while retracing the path (step 4 section 2.2). That this can not be expected to be otherwise, can be proved using the theory of Automata (59).

There is a problem in identifying the path of approach while tracing back, once it has entered into a loop for which the starting point does not belong to the loop. The entry in TEMP of Table A corresponding to node N1 or N2 (Fig. 2.6) gets modified. This difficulty can be overcome by having an additional column, called MASTER temporary. The MASTER temporary keeps track of the main logic flow and the other TEMP keeps track of paths due to loops. This will be further discussed in the next chapter which deals with the implementation of this algorithm.

A small such program and decision table corresponding to it are illustrated in Fig. 2.7 and Fig. 2.8.

2.5 Parsing:

A common problem in applying decision tables to non-trivial tasks is that the tables become so large that several of their main advantages are liable to be lost. It is useful in such cases to reorganise them into smaller tables by isolating the conditions, and the decision rules into independent groups, that is, to reduce a large decision table to a workable size while still retaining logical completeness.

The idea of parsing is illustrated here with the help of a limited entry decision table shown in Fig. 2.9.

C1	Y	_	_	_	N	_
C2	_	N	_	Y	_	Y
C3	Y	_	N	_	N	_
C4	_	Y	_	N	_	Y
C5	Y	_	N	_	Y	_
A	1	2	3	4	5	6

Fig. 2.9 Decision Table to be Parsed

Pollack (68) and Muthukrishnan (61) have given in detail procedures for formulation of decision tables. A brief summary is given below.

2.51 Assumptions in the formulation of Decision Tables:

The following restrictions are to be observed in formulating decision tables.

i) The conditions in a table should not imply any ordering regarding the sequence in which they are tested. Any sequence regarding the order of tests is expressed in the form of linked tables.

ii) There exists no prescribed order regarding the sequence in which the rules are arranged in a table.

iii) The set of rules collectively exhaust all combinations of conditions logically possible. This is automatically satisfied

when the ELSE rule is specified in a table.

Assumptions (i) and (ii) permit permutation of condition rows and rule columns, without altering the logic in a decision table. It is to be noted, however, that since the actions are executed in the sequence in which they are entered, action entries can not be permuted.

Assumptions (i) and (ii) given before allow rearrangement of rules and conditions. One such rearrangement of conditions and rules for Fig. 2.9 is shown in Fig. 2.10.

C1	Y	—	N	—	—	—
C5	Y	N	Y	—	—	—
C3	Y	N	N	—	—	—
C4	—	—	—	Y	N	Y
C2	—	—	—	N	Y	Y
A	1	2	3	4	5	6

Fig. 2.10 Parsed Table obtained from Fig.2.9.

The parsability of a decision table depends on the sparseness of its condition entry matrix. From this point of view, it is desirable to simplify the decision rules in a decision table and introduce 'don't care' entries wherever logically possible. There are two aspects of simplification: (i) simplification based on the semantics of the process described by a decision table and

(ii) simplification by applying some procedure based on the condition entries. The first aspect is not, in general, amenable to machine processing and calls for manual processing by the user.

Ned Chapin (8) defines this aspect of simplification as 'parsing' and describes some outlines for constructing decision table for complex logical procedures. The second aspect has been considered by Muthukrishnan (61).

Muthukrishnan's procedures are simple and straightforward. The implementation (Section 3.31) is based on his procedure.

The next chapter gives the details of implementation and a few sample examples with the corresponding decision tables are also included there.

CHAPTER III

IMPLEMENTATION

3.1 Introduction:

The algorithm described in the last chapter has been implemented using FORTRAN-IV language. A list processing language would be more suitable but FORTRAN-IV was chosen due to its wider availability and better I/O format.

The implementation of LOGITRAN (Logic Translator) is in two phases. In phase 1 blanks are squeezed out and the two tables A and B are formed. Also subtables are generated corresponding to complex logical IFs. In phase 2, tables A, B and sub-tables are used and the algorithm implemented. Duplicate entries of the same condition in the condition stub part is avoided as far as possible. Merging of sub-tables in the main decision table is also acomplished. Computed GOTO and Assigned GOTO statements are also allowed in the implementation. The user is expected to ensure that the program is syntactically correct, there are no missing and duplicate statement numbers and there is at least one occurence of a STOP or CALL EXIT or RETURN statement.

The output given to the user consists of a listing of his program, tables A and B, subtables and the decision table. He may suppress table A or table B by using appropriate control card options. The decision table would be parsed if requested.

The program is written completely in FORTRAN-IV and does

not use even a single sub-routine written in assembly language. Excepting for memory capacity, back up storage (tapes) etc. the program does not use any machine dependent features. However, it does take care of certain short comings of the IBM implementation of FORTRAN-IV on IBM 7044. An example would make this point clear:

```
        GO TO 10
     10 FORMAT (* THIS IS A BAD NEWS*)
```

Since FORMATS are non-executable statements (10), GO TO 10 is a logical mistake. But such mistakes are not pointed out by the IBM FORTRAN-IV compiler available at IIT/K. The present implementation takes care of this.

Comment cards have been used very liberally, functions of the various pointers discussed in detail and hints given to the users who would like to either understand or modify this implementation program.

Various segments of the program are discussed in the next and subsequent sections.

3.2 Phase 1:

The input to this phase is the users deck and output consists of tables A and B, and subtables for complex logical IF s, computed and Assigned GOTO s. After squeezing out the blanks from a statement, it does the classification of the statements into the following types:-

1. GOTO n
2. STOP, RETURN, CALL EXIT

3. Computed and Assigned GOTO

4. IF

5. All the rest

IF s are further classified and referenced by 'TYPE' as follows:

IF(XXX.OP.XXX)	GOTO nnn	TYPE 1
IF(XXX-XXX)	n1,n2,n3	TYPE 2
IF(XXX)	n1,n2,n3	TYPE 3
IF(XXX.OP.XXX)	Expression	TYPE 4
IF(XXX.OP.XXX)	CALL SUB	TYPE 5
IF(XXX.OP.XXX.AND.XXX.OP.XXX.---)XXX		LOGICAL
IF(XXX.AND..NOT.XXX)XXX		Containing
IF(.NOT.XXX)		declared
IF(XXX)XXX		logical
IF(XXX.GT.XXX.AND.XXX)XXX		variables.

The above classification has been done only from the point of view of ease of programming.

Classification of statements is done essentially on the basis of string and pattern matching. FORTRAN-IV does not have built in character handling instructions. Instead of going to assembly language, routines written in FORTRAN-IV were developed. The same are discussed in section 3.31.

In the case of complex logical IFs and computed and

assigned GOTOs there is a provision for upto 10 rules and 10 conditions. Miller's (57) thesis that most human beings can, at the best, keep track of about 7 ± 2 items at a time, has influenced the selection of the number 10.

Phase 1 assumes that the program* is syntactically correct. However, if that is not so, the implementation takes care of it by making CARYON zero and printing a message "THE FOLLOWING STATEMENT SEEMS TO BE WRONG IN SYNTAX OR IT HAS NOT BEEN TAKEN CARE OF IN THIS IMPLEMENTATION".

In the listing of the processer, given in Appendix III, phase 1, doing the classification of the statement, is identified by CLASFY.

Phase 2:

Phase 2 is the implementation of the actual algorithm. It takes care of conditions becoming duplicate, and calls MERGE to combine the subtables, corresponding to logical IFs with the main decision table.

Phase 2 can also call a subroutine PARSE which does the parsing of the decision table.

A brief description about the various subroutines which are called by LOGITRAN are given in the next section.

---

*Program, wherever mentioned, means users deck serving as input data to this processor. It is also some times called as SOURCE program.

3.31 Subroutines*:

BLOCK: This is the BLOCK DATA subprogram and helps in entering data into labelled COMMON Blocks (see section 3.32).

REMLBL: (REM ove L eft BL ank) It helps in removing the impertinent blanks on the left of nonblank characters in a word. This subroutine can in fact remove the occurence of any CHARacter starting from the left

(CHAR)

Ex. ƀ

ƀƀ123ƀ ⟹ 123ƀƀƀ

0

000IIT ⟹ IITƀƀƀ

BINBCD and BCDBIN: These subroutines are used to convert from BINary to BCD and vice versa. They are used for specifying a FORMAT for an input/output list during execution of the program. They are also used in string matching routines.

SQUEZE: This removes blanks from a statement, unless the same are a part and parcel of the statement as in FORMAT and DATA declarations. It normally does it for 80 columns but can be used to take care of upto 660 columns (one initial and 9 continuation cards, each having 7-72 columns).

SEARCH: This is used to SEARCH for a delimiter e.g. - , ( ) etc. etc. In case the search is successful FOUND=100 is returned, along with the character string, forming a word or part thereof, upto the delimiter.

---

*Block capital letters wherever given, are the actual variable names used in the program.

<u>Ex.</u> CALL SEARCH (MINUS, ALNUM,36,TERM1)

where MINUS = "-" a delimiter

ALNUM = AL phabets, NUM bers

36 = Size of the ALNUM

TERM1 = Returned word which is result of search.

An actual example will make it clear: Supposing we want to get the character string between (and - in the statement

```
        (7)  (10)
         IF ( IITK - ELEC ) 10,11,12
```

the starting character is in column 10, so

INITAL=10, is <u>linked</u> by labelled COMMON SOURCE, then

```
CALL SEARCH (MINUS, ALNUM, 36, TERMI)
```

will return from SEARCH

```
TERMI = IITK
```

This subroutine also returns a FINAL value, giving the column count for the last character before the delimiter. In the example cited above,

FINAL = 13.

<u>NODLMT</u>: (<u>NO</u> <u>D</u>e<u>L</u>i<u>M</u>i <u>T</u>er).

In statements where there is no delimiter explicitly appearing, we call this subroutine. It searches for a quantity OBTAIN which is followed by a character whose nature (alphabetic or numeric) serves as a "delimiter". An example will make it clear:

```
10   DO   100  IJ = 1,200,2
```

Now we donot have a special character as a delimiter between 100 (scope of DO) and the index IJ. We are able to distinguish between 100 and IJ because the first character of index must be an alphabet. NODLMT deals with such cases by the statement:

CALL NODLMT (NXTQAN,KK,PREVCH,KL,OBTAIN)

where OBTAIN = character string we are looking for, e.g. 100 in this case

PREVCH = Nature of characters in OBTAIN, numeric in this case

KL = Arry size of PREVCH

NXTQAN = Array of character following OBTAIN

KK = Array size of NXTQAN

It returns FOUND=100 if the search is successful.

It would be interesting to note that we donot have to call this subroutine to get the GOAL of a GOTO; for example for 300 of

GOTO 300

We call SEARCH, looking for a BLANK.

SEQUNC: (SEQUeNCe of delimiters). For a given string of characters (BUFFER) ,it gives the list of delimiters in the order as they appear in BUFFER, scanning from left to right.

Ex. CALL SEQUNC (LIST,M,JPREV)

where LIST = List of M delimiters in BUFFER found between columns INITAL and JPREV

This subroutine has been found of immense help in cases where calling SEARCH did not serve any fruitful purpose. This subroutine provides a mask and helps in getting the required quantity by the next SEARCH. In other words, the mask and the tree methods can be combined.

GETENT: (GET ENTry) This subroutine is used for IFs, computed and assigned GOTOs to get the character set within the matching pair of paranthesis, that is to get a well formed character string. It assumes that the last character encountered is a (and the next character of BUFFER is at INITAL (INITIAL). Before a RETURN from this subroutine, the last character (excluding ')') is at FINAL. It also returns MAXNOP, which is the number of matching pairs. If UPTO column 80 of BUFFER, the matching right paranthesis is not found, it returns

FOUND=0 ,

giving an indication that a continuation card is to be read, where in the matching pair may be found.

PUTBRN: (PUT BRaNches for an IF).

Having found out the relational operator (.LE.,.GT., etc. etc.) and the GOAL for true outcome of a logical IF, this subroutine puts the proper values for the three branches BR1, BR2, BR3, to be entered into table A. It returns FOUND=100 in case the operator met with is a logical (AND,OR,NOT) rather than a relational; it returns FOUND as 1, 2, and 3 for AND, OR and NOT respectively.

Since for Logical IFs , the branch for 'false' condition is implicit, PUTBRN inserts XXXXX (STARS) for the corresponding branches.

DTENT : (Decision Table ENTrys). This subroutine helps in getting the entrys for the rule next to the current one. Corresponding to any tree or two branch flowchart, if we get the decision table, then in general, any two adjacent rules are found to differ usually in one entry. Thus the condition entries for the next rule can be "mapped" from the current rule. But there are rules for which this mapping will not represent the actual entries unless proper care is taken. There are instances when a certain impertinent condition will be shown as pertinent unless taken care of. DTENT, with the help of PREVNO (this is same as TEMP of the last chapter) helps in back tracking the tree from STOP/RETURN back towards start (BEGIN), helping in retaining only the pertinent of the entries got by straight mapping and deleting the impertinent ones. DTENT returns a value 1 for FINAL for tree structured and 2 for non-tree structured programs (section 2.4). ('Final' has no significance here. This pointer is used only to economise in memory). In the latter case of FINAL=2, DTENT is called again with one of the arguments as MASTER instead of PREVNO. MASTER keeps track of the main logic flow and PREVNO of the paths due to loops etc. etc.

MERGE : It is called when an entry corresponding to a subtable is found in table A. This subroutine helps in avoiding duplication of condition stubs of subtables corresponding to logical IFs and merge

the various subtables with the main decision table. It also avoids duplication of subtables.

UNIQUE Subroutine UNIQUE compares elements of array LONGEX with those of COLS; if a new element is found in LONGEX, it is appended to COLS; if an element is found to be repeated, nothing is done. Thus elements of COLS are a subset of the elements of LONGEX with the property, that each element of the latter appears in the former but only once. This subroutine helps in establishing the proper linkages between the various subtables and the main table. An action such as given below:

...	..	..	..
GOTO	AS10	..	..

where AS10 is the header for a subtable, would require that AS10 be explicitly printed in the output. We would not require to output a subtable again if it has already been printed once. This needs a list of elements forming COLS, all the elements being unique.

PARSE It does the parsing of a large decision table, and is based on the algorithm referred to in section 2.5 of last chapter. Calling PARSE is optional.

This subroutine in turn calls CMPL, LOWEST and REPLAC. The decision table to be parsed is first stored on tape for reference, its entries are converted to boolean form and parsing done. The original decision table is retrieved from tape and the various

conditions, rules, actions permuted, dictated by the parsed table.

This can be used independently of other subroutines if the object is only to parse a large decision table. The options to be used for this are given in detail in (87) and in Appendix V.

COMPL: This subroutine returns the 1's compliment of an array of 1's and 0's. It is called by PARSE and can handle any row (WHICH) of an array (EXENT) and return RESULT.

LOWEST: This subroutine is also called by PARSE and picks the lowest element of a set of numbers. Further, it returns the position of the lowest element. If there are two elements, equal in magnitude and lowest, it returns the position corresponding to the one which appears earlier in the set.

REPLAC: Subroutine REPLACe interchanges two rows of an array. Since we would like it to do it for arrays of varying sizes, the dimensions of the array, WHICH is to be replaced WHERE and where should we PUT the latter, are linked by arguments. This subroutine helps in permuting rows of a decision table. We call REPLAC first for permuting two condition stubs and then to permute the corresponding condition entries of the decision table.

REPCOL: For parsing, having permuted the rows, we have to permute the columns (rules) also. For this we would require a subprogram which would be similar to REPLAC, but should handle columns instead of rows. There are two alternatives: take a transpose and call REPLAC or write a subprogram to handle columns directly. Both were tried,

and it was felt that the second method had a slight edge over the first in so far as the memory was concerned. The outcome was REPCOL (REPlace COLumn).

3.32 Common Blocks:

For economy of memory, variables including array names are declared common. Common declaration provides a way to allocate the same memory space to variables of two are more different programs. A number of common blocks have been named in this program. A list of common blocks along with their lengths and an idea about their appearance in the various subprograms is given in detail in the listing of the main program. The idea is to facilitate changes, if any, to be made at a later stage. A brief discussion of these blocks follows.

Blank Common // This common contains variables which are used mainly in Phase 2 and pertain to the various parts of the decision table.

RULES It provides the linkage for subtables in the different subprograms.

SOURCE As the name implies, it links variables which have some thing to do with the scanning or pattern matching of the SOURCE* program.

DETAB Variables like BEGIN, GOTO,STOP,STARS(XXXXX) which have values, that are the same as implied by their names are used at a number of

---

*See footnote on page 36.

places in all the subprograms. These contain DATA and are linked by BLOCK DATA subprogram.

LOOKUP It was found time saving to use an array BCD TABle to store the BCD equivalents of the serial numbers (BINary) of the SOURCE program statements. The implementation requires pattern matching and for this a logical comparison of the string patterns was found to be better than arithmetic comparison. Hence the need for B.C.D. equivalents, kept in BCDTAB. A LOOKUP technique is thus time saving as compared to calculating the B.C.D. equivalent every time it is needed.

3.4 LANGUAGE FOR IMPLEMENTATION:

FORTRAN-IV has been used for implementing the alogirthm discussed in the last chapter. Which would have been the best language for this purpose, is an important aspect to go into.

From a brief description of some of the subroutines given earlier in section 3.31, it would seem that the implementing language should be capable of doing string manipulation. The choice, thus, would fall on an assembly language or a higher level language, specially suited for character handling. However, for Phase 2, wherein the actual algorithm is implemented, a list processing language would seem to be more suitable.

SNOBOL is good at string manipulation but the compiler for that available at IITK is very slow since it produces a very inefficient object code. Further this language will be good for

Phase 1 but not for Phase 2. For Phase 2, a list processing language would be more suitable. LISP would have been ideal but is not available at IIT/Kanpur.

FORTRAN-IV was chosen due to its wider availability. It is well defined and runs on both our second and third generation computers. It has deficiencies but they are known. It has a fare amount of machine independence.(76).

3.5 Restrictions and Conventions:

The user is not expected to observe any special restrictions. The only precaution expected of him is that the program is syntax checked and there are no missing or duplicate statement numbers. The idea is that the user, making use of this processor, should not be required to make any changes in his deck as far as possible. No user would like to make changes in his deck or to put in any extra effort if he is interested in only getting the logic of the program.

However, the algorithm expects at least one occurence of any one of STOP, RETURN or CALL EXIT statements. A very valid question likely to be asked is 'what about the programs which donot include any of the above statements'. Let us consider the program skelton given in Fig.3.1. We read a set of data, do some computation, test

```
            .
            :
      5     READ 6, Data
            :
            IF( XXX   ) 9,10,11
            :
      9     ....
            .
            .
            GOTO 5
     10     .....
            .
            .
            GOTO 5
     11     .....
            .
            .
            GOTO 5
            .....
            .
            .
```

Fig. 3.1 Skeleton of a Program:

a condition, some more computation, print it and read the next data card, repeat the above and so on. When do we stop? Well! Somebody answers! 'Till all the data cards are read' i.e. till the next $JOB card is encountered. The author regards it as bad programming: the user is depending too much on the system supervisor. Such cases are not allowed by this processor.

Implementation, in its present form can take care of 32

conditions and 32 rules for the main decision table and 10 rules and 10 conditions for sub-decision tables. However, one can introduce changes to suit individual needs, very easily because comment cards have been used very liberally at various places which not only serve as documentation, but would also facilitate updating.

3.6 Machine Dependent Features:

The FORTRAN-IV to Decision Table translator LOGITRAN has been written and implemented on IBM 7044. The endeavour has been to avoid the machine dependent features, as far as possible, so that the program could be run on other machines on which FORTRAN-IV is available. In fact this was the reason which prompted the author to avoid subroutines written in assembly language. The translator LOGITRAN, however, does include a few minor machine dependent, features, which are given below.

MEMORY: On IBM 7044 of IIT/Kanpur, the core memory available is 32,768 words. FORTRAN users can also use tape units 0,1,2,3 and 4 for back up storage. LOGITRAN has been written with the above memory in view. For machines of smaller memory, the sizes of arrays appearing in blank (//) COMMON and LOOKUP can be reduced to suit the available space. The other alternative is to over lay the program by using the LINK and CHAIN facility.

TAPE 99: Logical unit 99 is 'similar' to logical units 0 to 4 which are attached to tape units. But the logical unit 99, or Tape 99 as it is usually called in IIT/K, has been created (locally) without

any physical input-output device attached to it. Tape 99 is not a standard feature but is available on most processors under one name or the other. In case such a facility with the same syntax, but under a different name, is available, by using a $NAME card (38), we can use the translator. Appendix II gives details about TAPE 99.

3.7 OUTPUT (EXAMPLES):

The output for the user consists of a listing of his program, tables A, B and subtables and the decision table. He may suppress table A or table B by using appropriate control card options.

A number of sample programs along with their corresponding output are given here (Fig. 3.2 to 3.4).* It has been found that decision tables obtained are easily readable and are very useful for logic checking. They are superior to flowcharts and their superiority is very evident in a tree structured program with a large number of branches.

Example Fig. 3.2 contains a partial decision table along with the program, clearly bringing out the utility of D.T.s in debugging. An example of a non-tree structured program has already been dealt in the last chapter. (Fig. 2.7, 2.8)

3.8 Conclusions and Suggestions for Improvement of Implementation:

Programs which generate decision tables corresponding to syntactically correct FORTRAN programs can be used for program

---

*See Appendix IV and Appendix VI.

```
C       PROGRAM 2- CODED IN FORTRAN IV SUBSET
C
C       CLEARLY POINTSOUT THE POTENTIALITY OF DECISION TABLES AS
C       A GOOD DEBUGGING AID
C
        READ 35,MPROD,MCUSTR,MORDER
35      FORMAT(3I3)
        IF(MPROD-2) 10,20,30
30      PRINT 40
40      FORMAT(1H ,*ERROR IN DATA*)
        STOP
20      IF(MCUSTR.EQ.1.AND.MORDER.EQ.1)  GOTO 50
        IF(MCUSTR.EQ.1.AND.MORDER.EQ.2)  GOTO 75
        IF(MCUSTR.EQ.1.AND.MORDER.EQ.3)  GOTO 100
        IF(MCUSTR.EQ.2.AND.MORDER.EQ.1)  GOTO 75
        IF(MCUSTR.EQ.2.AND.MORDER.EQ.2)  GOTO 100
        IF(MCUSTR.EQ.2.AND.MORDER.EQ.3)  GOTO 125
        IF(MCUSTR.EQ.3)  GOTO 75
C
C
10      IF(MPROD.EQ.1)  GOTO 50
        GOTO 30
50      DISCT=0.05
        GOTO 1000
75      DISCT=0.075
        GOTO 1000
100     DISCT=0.10
        GOTO 1000
125     DISCT=0.125
1000    PRICE=ORDER*(1.-DISCT)
        PRINT 25,MPROD,MCUSTR,PRICE
25      FORMAT(1H ,I2,2X,I2,2X,F9.2)
C
        STOP
        END
```

DECISION TABLES

ENTRY CONDITION	RULE 1	RULE 2	RULE 3	RULE 4	RULE 5	RULE 6
MPROD - 2	.LT.0	.LT.0	.EQ.0	.EQ.0	.EQ.0	
MPROD -'1	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	
MCUSTR -1			.NE.0	.NE.0	.NE.0	
MCUSTR -2			.NE.0	.NE.0	.NE.0	
MCUSTR -3			.NE.0	.NE.0	.NE.0	
MORDER -3						

ACTIONS

Fig. 3.2 Partial Decision Table, Pinpointing Mistakes in Logic.

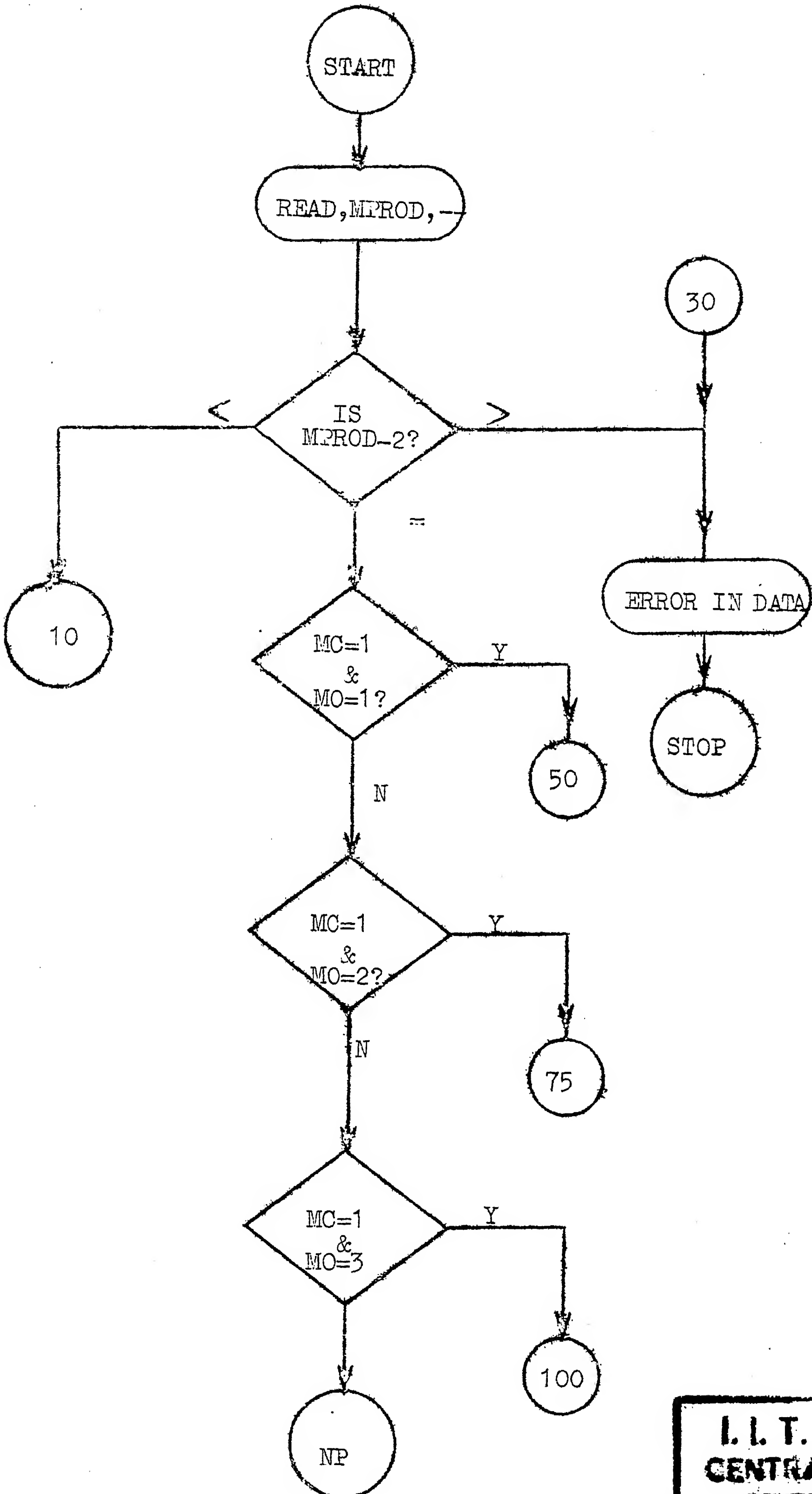


Fig. 3.21. FLOWCHART for Figure 3.2. Continued on next page.

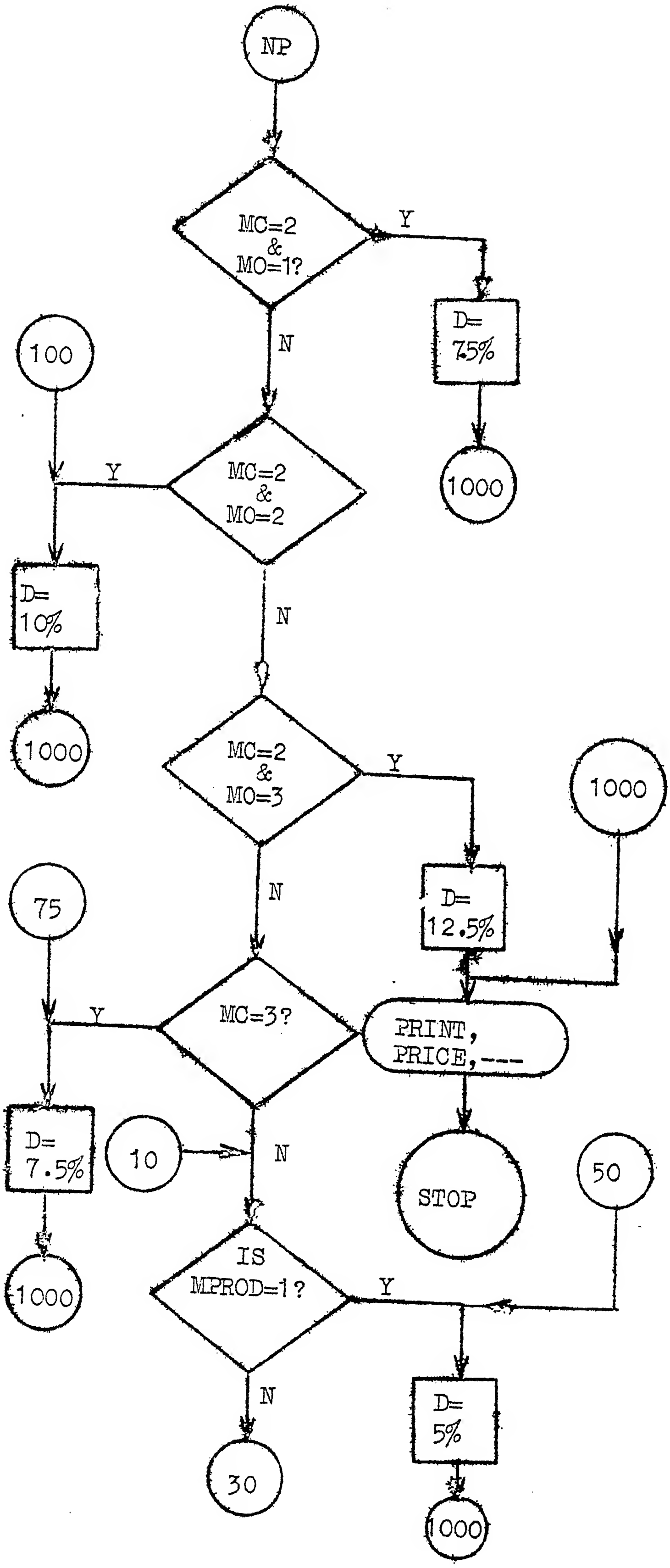


Fig. 3.21. <u>Continued from last page.</u>

```
C       PROGRAM 3 BY DR.V.R.       AS MODIFIED FOR LOGICAL ORS
        READ 35,MPROD,MCUSTR,MORDER
35      FORMAT(3I3)
        IF(MPROD-2) 10,20,30
30      PRINT 40
40      FORMAT(1H ,*ERROR IN DATA*)
        STOP
20      IF(MCUSTR.EQ.1.AND.MORDER.EQ.1)  GO TO 50
        IF(MCUSTR.EQ.1.AND.MORDER.EQ.2.OR.MCUSTR.EQ.2.AND.MORDER
       1.EQ.1.OR.MCUSTR.EQ.3)  GOTO 75
        IF(MCUSTR.EQ.1.AND.MORDER.EQ.3.OR.MCUSTR.EQ.2.AND.MORDER.
       1EQ.2)  GOTO 100
C
        IF(MCUSTR.EQ.2.AND.MORDER.EQ.1)  GOTO 125
10      IF(MPROD.EQ.1)  GO TO 50
        GOTO 30
50      DISCT=0.05
        GO TO 1000
75      DISCT=0.075
        GO TO 1000
25      FORMAT(1H ,I1,2X,I1,2X,F9.2)
100     DISCT=0.10
        GO TO 1000
125     DISCT=0.125
1000    PRICE=ORDER*(1.-DISCT)
        PRINT 25,MPROD,MCUSTR,PRICE
        STOP
        END
```

Fig. 3.3 Sample Program.

DECISION TABLE

ENTRY CONDITION	RULE 1	2	3	4	5	6	7	8	9	10	11	12
MPROD-2	.LT.0	.LT.0	.EQ.0	.EQ.0	.EQ.0	.EQ.0	.EQ.0	.EQ.0	.EQ.0	.EQ.0	.EQ.0	.GT.0
MPROD-1	.NE.0	.EQ.0	.NE.0	.EQ.0								
MCUSTR-2					.EQ.0		.EQ.0		.EQ.0			
MORDER-1					.EQ.0				.EQ.0		.EQ.0	
MCUSTR-1						.EQ.0		.EQ.0			.EQ.0	
MORDER-3						.EQ.0						
MORDER-2							.EQ.0	.EQ.0				
MCUSTR-3										.EQ.0		
ACTIONS												
GO TO	30	50	30	50	125	100	100	75	75	75	50	30
GO TO	STOP	1000	STOP	1000	STOP	1000	1000	1000	1000	1000	1000	STOP
GO TO		STOP		STOP		STOP	STOP	STOP	STOP	STOP	STOP	

Fig. 3.4 Decision Table for Program Fig. 3.3.

documentation (as a replacement of flowcharts) and also as diagnostic aids for semantic or logical checking of programs. The latter, it is felt, would be very useful particularly because of the easy readability of decision tables (Fig. 3.2).

It must be emphasised at the outset, that the endeavour has been to test the feasibility of automated conversion of programs to decision tables and to see how far they help in documentation and in debugging of programs. This is an area in which the potentiality of decision tables had not been explored so far. The present venture is a beginning in this direction, and hence by no means 'a last word' or an efficient outcome is claimed. Implementation wise also there is a good scope for improvement.

1. Phase 1 of the implementation which scans the source program, statement by statement, makes use of TREE technique for branching. MASKING is another technique which is used in such situations. Both the techniques are used. Instead of testing for a delimiter one by one, we can use subroutine PATTERN, thereby (Appendix III, page 184) testing for a pattern of delimiters. The advantage is that classification of statements and the various types of IFs (section 3.2) can be done more efficiently.

2. Subroutine PARSE does not pack information about the condition entries into bits; it uses words. When bits are used, a considerable space in memory can be saved. The reason, this was not done in the present implementation was, that bit handling would

require the use of the machine dependent features: non standard AND, OR, COMPL functions or the use of subroutines written in assembly language.

3. At a number of places records have been written on tape in BCD format and when needed are read also in BCD format, This has been done to facilitate modifications, if any, at a latter date. The idea was that if one wants to make some changes, one should not get lost. However, if this aspect is not to be considered, then binary READ and WRITE instructions would do. This change would result in faster processing besides saving in memory.

4. The processer is written in FORTRAN-IV. We should try to use D.T.s for writing a program for D.T.s , that is, use a processor (61) which allows decision tables as a direct input. D.T.s embeded in FORTRAN should go a long way in improving the over all efficiency of the processor.

The above discussion was for this implementation of the processor. However, we must answer a few very basic fundamental questions:

1) Is the algorithm the best?

2) Does the output depict all the different types of bugs in a FORTRAN program?

3) Can we apply a similar algorithm for programs written in languages other than FORTRAN?

An attempt will not be made to give in detail the answer to all these questions in this chapter, because the same are discussed

at length in the subsequent chapters of this thesis. However, a brief discussion would not be out of place. Let us take up each question one by one.

It is not claimed that the algorithm is the last word. In fact, it has already been mentioned that it is just a beginning. The use of decision tables as an aid in debugging had not been explored so far. Excepting Miller and Maloney (58) nobody, to the best of the knowledge of the author, had even thought of using D.T.s for debugging. Even these two authors discarded D.T.s for debugging for programs other than those written in tabular language. It is, thus, just a beginning in this area and there is much scope for improvement, which will be clear, even as we go along and answer the second question.

Well, certainly the algorithm in its present from does not throw any light on endless loops and program segments which are never reached for. The considerations mentioned just now, along with an offshoot of the first question namely 'can we have a binary procedure for conversion of programs to D.T.s' has led to the algorithm given in the next chapter of this thesis, where it is discussed in detail.

Lastly, it is felt that the philosophy of the algorithm will work for languages other than FORTRAN also, though the details of how the algorithm could be implemented would depend upon the particular language under consideration. Conversion of assembly

language programs to decision table is discussed in chapter V of this thesis. There, it is dealt at length along with the various other difficulties encountered in conversion.

An altogether different approach would be to develop a general processor which could take care of all languages: let one describe the syntax of the control statements in his language, in a meta language (85) and then for any program, it would be possible to get a decision table.

Before we pass on to the next chapter let us try to answer another question: Is it possible to get a decision table for a program which modifies itself? (19,41,92) NO! We can not do it. Fortunately, FORTRAN does not allow program self-modification, and so the algorithm, even in its present form, does not have to bother about it.

CHAPTER IV

DECISION TABLES
FOR
FORTRAN PROGRAMS: MATRIX DESCRIPTION

4.1 Motivation:

Towards the end of the last chapter a few questions had been posed whether decision tables can give us an idea about endless loops etc. etc. Decision Tables, by themselves donot give any such indication. But, as was mentioned in Section 1.31, the Boolean matrices associated with a program can. The main advantages of these matrices are that, besides being suitable for machine computation, (30,83) they can readily give us an idea about blocks which:

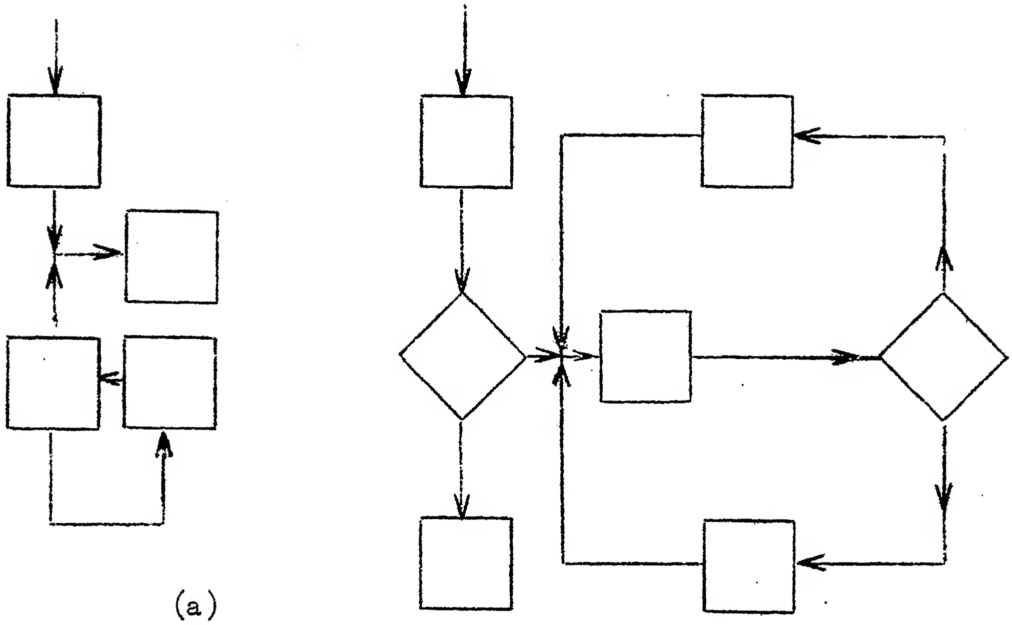
i) can not be reached from input (Fig. 4.1a),

ii) do not have exits (Fig. 4.1b),

iii) are in loops (Fig. 4.1c),

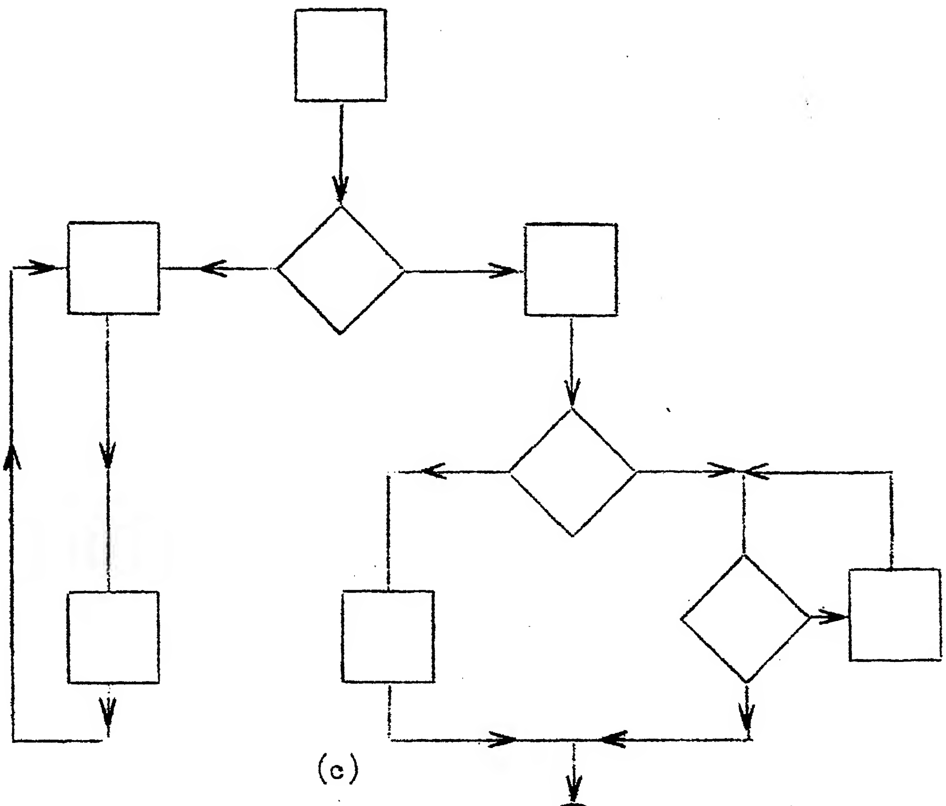
iv) decompose into to form sub-programs.

It will be noted that, excepting matrices, no other existing representation can give us any idea about loops. Decision Tables, though very good in many other respects, completely fail in this respect, unless a detailed analysis is undertaken of the condition stub, the action stub and the corresponding entries. Thus it seems that there are certain features of decision tables and certain features of Boolean matrices which make these representations potential debugging aids. Moreover their potentialities are



(a)

(b)



(c)

For the sake of completeness, some of the definitions given by him are given below:

Steps: nodes or points of the graph, or items of the matrix.

Symbol: $<$ - means precedes, $a < b$ means a precedes b

Chain: a sequence of relationships of the form

$$a_1 < a_2, \; a_2 < a_3, \; a_3 < a_4, \text{---}, a_{n-1} < a_n$$

which can be written as

$$a_1 < a_2 < a_3 < a_4 \text{ - - - - - - - - } < a_n$$

Implication: -If $a < b$ and $b < c$,

then $a < c$

is known as the implication of relations $a < b$, $b < c$

The physical significance of raising the matrix to its higher powers (71) can now be seen. Examining the graph given in Fig. 4.2 it will be seen that implications of the chain are 15,28 and 18.

Now, the rule for finding the implication of a chain of two elements, namely that $b_{ij} = 1$ and $b_{jk} = 1$, then $b_{ik} = 1$, corresponds to the rule for finding the square of a Boolean matrix. Thus finding implications is equivalent to raising to higher powers. Raising to higher and higher powers yields the implications of the longer sub-chains, until finally some power of the matrix has all null elements (if the matrix is consistent), indicating that all the

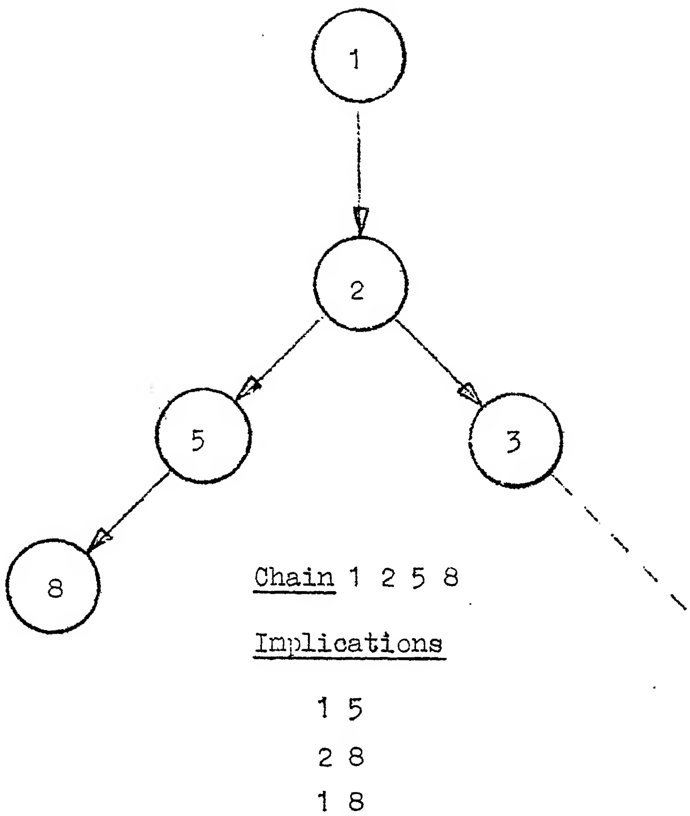


Fig. 4.2 Chain and Implications of a Directed Graph.

implications have been made explicit. As already mentioned, Merimonts procedure, besides giving us a better physical picture, eliminates much tedious computation.

Contradictions: The matrix of a set of items in a closed loop will contain a non-zero element in every column and every row. i.e. there will be no ZERO rows or ZERO COLUMNS. Since the matrix of a sub-set of the original set of items will be a principal sub-matrix of the original matrix, the criterion can be stated thus: A set of precedence relations is consistent, if and only if, every principal sub-matrix has at least one zero row or zero column.

4.21 Procedure for Searching for Contradictions:

We delete, successively, from the matrix any items which have either zero rows or columns i.e. all exits and entrance. This will yield a sub-matrix, which now has new exits and entrances. The problem is repeated until either every item has been deleted (for the consistent set) or a sub-matrix with no zero rows or columns remains for the inconsistent set.

4.3 Limitations of Matrix Representation:

A limitation of use of the Boolean matrix, (connectivity), when considering decision elements only, is that two different paths from one decision element to the next one can not be illustrated. An example will make it clear (Fig. 4.3). In the matrix all that is likely to be shown for these two paths is a single 1 in (6,7) (in row for D6 and column for D7.) Karp (41) has suggested a modification

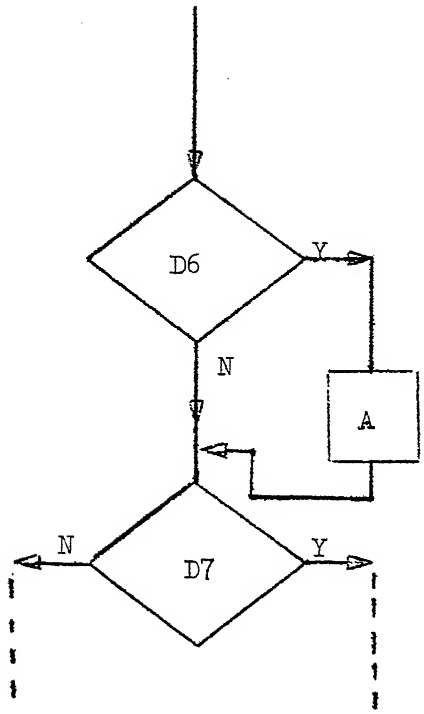


Fig. 4.3

for such cases. A refinement of the Boolean matrix can be accomplished by replacing the 1's by the notation of the actual path followed. In the connectivity matrix, the entry in row D6, column D7 could be $6 \vee \overline{6}$ or $6+\overline{6}$, $\vee$ symbolising the logical OR, indicating that the program can arrive at D7 by one of the two alternative paths.

The above will take care of binary decision elements: what about an IF statement of FORTRAN-II which allows upto 3 branches? Thus, not only we must keep provision for more than 2 way branching, for future processing, we should think of some representation which is better than the connectivity matrix. The author proposes a new matrix called, STRUCTURE matrix for this.

4.4 Structure Matrix:

The structure of a program is usually determined by a detailed specification describing the program, and can have various types of representations, some of which have been discussed in previous chapters. Such a representation should have the following properties:

i) It should be easy to construct and reproduce.

ii) It should be adaptable to handling by machine.

iii) It should contain all of the information provided by program.

Prosser has also listed similar properties. In the 3rd property he has used 'topology of the flow diagram' instead of

'program', and then he suggests the connectivity (Boolean) matrix as "a representation which has all these properties". The contention of the author is that since YES, NO, or $<, =, >$ branch paths are a part and parcel of the flowcharts, unless the representation includes some way of depicting it, in some unambiguous manner, the representation is not good. The desirable features listed by Prosser are not satisfied by a connectivity matrix. It seems, he missed this aspect. The STRUCTURE matrix is proposed for this.

The structure matrix will have the various elements as given by the following decision table (Fig. 4.4). It may be noted that this table has been drawn, keeping in view FORTRAN, and for any other language, minor changes may have to be made in the D.T.

STATEMENT BELONGS TO SET	Y	N	N	N	N
STATEMENT IS UNCONDITIONAL TRANSFER	N	Y	N	N	N
CONDITIONAL TRANSFER	N	N	Y	N	N
DECLARATION (FORMAT, DIMENSION, END)	N	N	N	Y	N
STOP/RETURN/CALL EXIT	N	N	N	N	Y
Element of STRUCTURE matrix is:	S	1	OP	N	ƀ

SET = I/O, Arithmetic Assignment, DO, CONTINUE

S stands for sequential

1 stands for the destination of GOTO

Relational OP stands for symbols like NE,EQ,GT,LE,GE,LT

N stands for 'do Not bother' (IgNore)

ƀ stands for blank i.e. donot enter anything.

Fig. 4.4 DECISION TABLE FOR 'S' MATRIX ELEMENTS.

```
C       STUDENT - HOMEWORK/PICNIC PROBLEM
        READ 10,HWORK,WETHR ,PENDNG,CLEAR
        IF(HWORK - PENDNG) 20,30,20
20      IF( WETHR - CLEAR) 40,50,40
50      PRINT 60
        STOP
40      PRINT 70
        STOP
30      PRINT 80
        GOTO 20
60      FORMAT( 12HGO ON PICNIC)
70      FORMAT( 12HGO TO TEMPLE)
80      FORMAT( 25HCOMPLETE HOMEWORK MORNING)
10      FORMAT(4A6)
        END
```

Fig. 4.5 Homework Picnic Problem.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1		S												
2			NE					EQ						
3				EQ		NE								
4					S									
5														
6							S							
7														
8									S					
9			1											
10										N				
11											N			
12												N		
13													N	
14														N

Fig. 4.6 Structure Matrix for Program of Fig. 4.5.

The advantages of, this type of entries for the elements of 'S' will be seen when we discuss the algorithm in section 4.8.

To illustrate, the STUDENT HOMEWORK/PICNIC problem, discussed in detail in Section 2.2 (Fig. 2.1), will be considered again (Fig. 4.5). It is to be noted that 'S' matrix contains N s as main diagonal elements for all declarations, that is FORMATS, type declarations etc. etc. The corresponding 'S' matrix is given in Fig. 4.6.

It may be noted that, whereas it is possible to arrive at the connectivity matrix (henceforth, called P matrix, after Prosser) from the 'S' matrix, without reference to the original program, the reverse is not possible. There in lies the advantage. Thus 'S' matrix is more general, conveys more information and will be more meaningful to us if we ultimately want to get the decision table for the program via matrices.

4.5 Matrices or Decision Tables:

The question is not 'shall we use matrices or should we go in for decision tables'?

The above section heading has been deliberately chosen. The choice is not to be made between the two: the two supplement each other. Whereas D.Ts can display the logic of the program in a concise elegant manner, the matrix representation can help us in locating endless loops, spurious set of statements and in finding

out if the program can be decomposed into relatively independent subprograms. The two representations should be utilized to supplement each other, as a whole providing a good effective debugging aid. We shall come to it again in section 4.9.

4.6 Machine Generation of 'S':

Before we pass on to the next section, where in, the algorithm for getting a decision table from a matrix is discussed, we must answer if 'S' matrix satisfies the other requirements as given in section 4.4, that is, 'it should be easy to construct and reproduce' and should be amenable to handling by machine. It does have this property.

In fact it could be said that, to some extent it has already been obtained by a program: only a few minor modifications introduced in phase 1 of the implementation, discussed in the last chapter, can give us the 'S' matrix. Once 'S' is available, converting it to a Boolean matrix is a straightforward procedure.

4.7 Structure Matrix to Connectivity Matrix:

The structure matrix for a program was discussed in previous sections. An example of such a matrix was given in Fig.4.6. In order to get an idea about loops, spurious blocks etc., one must convert it to a Boolean matrix. This can be done by the procedure, given on the next page:

	1	2	3	4	5	6	7	8	9
1		1							
2			1					1	
3				1		1			
4					1				
5									
6							1		
7									
8									9
9			1						

Fig. 4.7 S' Matrix.

i) Replace every S by a 1.

ii) Replace all relational operators by a 1.

iii) Delete ROWS and COLUMNS which contain an N as a diagonal element.

iv) Retain other elements as they are, that is, 1 for 1 and blank (or zero) for a blank (or zero).

With the above changes introduced, the connectivity matrix for the structure matrix of Fig. 4.6 is given in Fig. 4.7.

4.8 Algorithm: Matrix to Decision Table:

In section 4.4 (Fig. 4.4) we have seen how we can get the entries of 'S' matrix, corresponding to various types of statements of a FORTRAN program. As has been mentioned earlier, the 'S' matrix contains information regarding the conditional and the unconditional transfer of controls besides giving an idea about computational steps in which the control is transferred to the next sequential statement. Precisely this is the information we need to get the structure of a program in the form of a decision table. It will be recalled that the algorithm discussed in chapter II requires tables A and B to be generated; the two tables depict the structure of the program. Since, the 'S' matrix itself, depicts the structure of the program, there is no need to separately generate tables A and B. The algorithm described below is a systematic procedure to trace the different paths of flow in a program. This will be further clarified as we go along describing it step wise.

To explain the converting algorithm we shall again use the example given in Fig. 2.1. While getting this structure matrix, when we come across an IF statement, we put down the condition part (character string between matching parenthesis) in the corresponding row on the left of the serial number. Call this <u>stub</u>. Column serial numbers have been given for the sake of convenience. The decision table is developed as follows:

1. Delete the rows and columns which have an N as a diagonal element. Call the remaining matrix as S' (Fig. 4.8).

2. Scan the first row of S' matrix and count the number of non-blank elements. Put down this number in an additional column called TOTAL. (column 15 in this case).

3a. Repeat step 2 for all rows of S'. The elements of column 15, rows 1 to 9 are respectively 1,2,2,1,0,1,0,1,1.

3b. Enter B for 'Begin' in column TEMP, row 1.

4. Look up S' (1,15) i.e. TOTAL (1). It will have a a value 1 or 2 or 3. (It can not have a TOTAL = 0 because that would mean STOP or RETURN as the first statement). If it is 1, go to step 5 otherwise go to step 8.

5. Pick up the non-blank entry of this row. See its column number (vertically up).

6. Take up the row whose serial number corresponds to the column number. Enter in TEMP column of this row, the row number which led us to it. If the non-blank entry is other than

STUB		1	2	3	4	5	6	7	8	9	TOTAL	TEMP	
	1		S								1	B	
HWORK-PENDNG	2			NE					EQ		2	1	
WETHR-CLEAR	3				EQ		NE				2	2	9
	4					S					1	3	
	5										0	4	
	6							S			1	3	
	7										0	6	
	8									S	1	2	
	9			1							1	8	

Fig. 4.8 S' Matrix with Stub, Total and Temp

	1	2	3	4
HWORK - PENDNG	NE	NE	EQ	EQ
WETHR - CLEAR	EQ	NE	EQ	NE
GO TO	4	6	8	8
GO TO	STOP	STOP	4	6
GO TO			STOP	STOP

Fig. 4.9 Decision Table

an S in one step right off-diagonal position* go to step 7 otherwise enter its number as an action entry and go to step 7.

7. Look up TOTAL. If it is 1 repeat step 5. If it is 0, go to step 9, if it is 2 or 3 go to step 8.

8a. Enter in TEMP for this row, the number of the row which led us to it.

8b. Pick up the characters from STUB column and enter it as condition stub of decision table. Scanning left to right, circle the first non-blank element. Enter it as condition entry in the decision table. Look vertically up. Get column number. Go to step 7. In case there are no more uncircled elements left in this row, go to step 11.

9. This corresponds to a STOP. Enter it as an action entry. The next rule is to be found.

Look up TEMP and take up the row whose serial number appears in it (TEMP).

10. Look up TOTAL. If it is 1 take up the row whose number appears in TEMP, otherwise go to step 8a.

---

* If the row under consideration is K^{th}, then main diagonal element is (K,K) and one step right, off - diagonal element is (K,K + 1). We call it ROD element.

11. Remove circles from the elements of this row. Next take up the row whose serial number appears in TEMP of this row. Repeat step 7, unless we have reached the row with lowest serial number, which has TEMP containing a 'B', when we can stop, because all paths of the tree have been considered.

12. In case at any stage we come accross an S and the next immediate higher serial number row (or column) is missing in S', we go to the still next higher. The missing row is because of deletion of N for non-executable statements.

S' matrix appended with STUB, TOTAL and TEMP columns is shown in Fig. 4.8 and decision table in Fig. 4.9.

The case of computed and assigned GOTOS and complex logical IFs can be treated in a manner similar to the one given in chapter II. We treat these as sub-decision tables and establish proper linkages, with the main table, each sub-decision table for a complex logical IF contributing an additional row and an additional column, and the corresponding entries.

4.9 MATDET: An Effective Debugging Scheme:

In section 4.4 it was mentioned that matrices and decision tables jointly help in debugging by pinpointing endless loops, redundant blocks, relatively independent subprograms and also provide an elegant display of logic in a tabular form. We call

this joint MATrices - DEcision Tables approach to debugging as MATDET. MATDET can be utilized to advantage in an actual set up, as follows (Fig. 4.10).

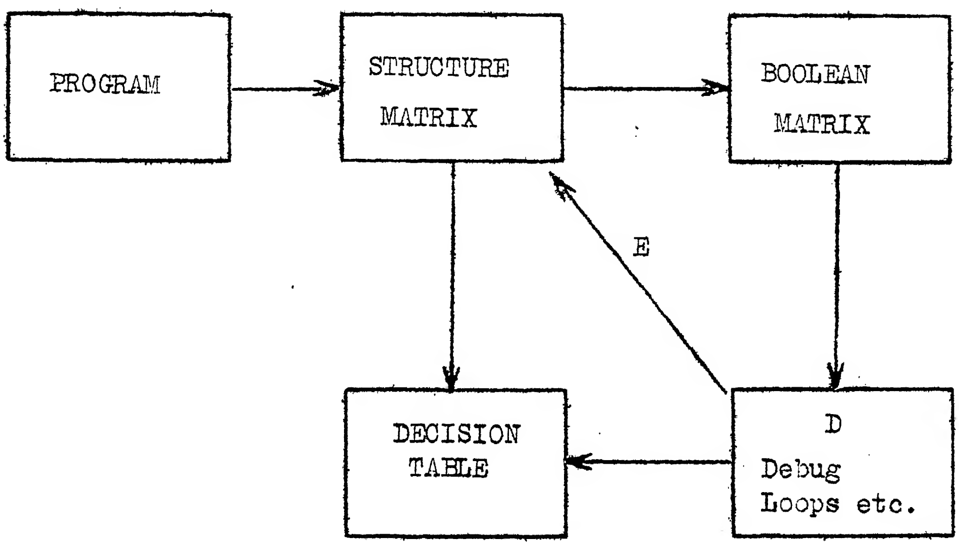


Fig. 4.10 MATDET: An Effective Debugging Scheme

From program we get the 'S' matrix. Convert 'S' matrix to a Boolean matrix and by the technique discussed in section 4.21 get information regarding loops, relatively independent sub-programs etc. etc. This information is very useful for long programs and can be utilized to 'parse' the program and get a set of linked decision tables. The arrow, marked E, indicates that the structure matrix, based upon the information got from block D, can be 'divided' into a number of sub-matrices to get the above mentioned set of linked decision tables.

CHAPTER V

CONVERSION OF ASSEMBLY LANGUAGE PROGRAMS TO DECISION TABLES

5.1 Introduction:

The obsolescence time of computers have been about 5 years and this has created the problem of running programs written for an old computer on a new one. Economy of investment in terms of time and money requires that this be done with a minimum of extra effort. Some of the techniques which aid in conversion of programs, written for one machine, to be run on another, as also the difficulties of translation of computer languages have been dealt before in section 1.32. In this chapter a new approach, where in decision tables are used as an intermediate step in the conversion process, is proposed.

Levels of Conversion:

The conversion processes may be limited to one of the following:

i) One assembly language to another.

ii) One assembly language to a higher level language.

iii) Object code to an assembly language (Deassemblers) (66,75).

iv) One higher level to another higher level e.g. FORTRAN-II to FORTRAN-IV or FORTRAN-IV to Decision Tables.

v) Higher level to lower levels e.g. compiler, assembler programs.

In the present context, our interest is only in (i) and (ii).

An algorithm for conversion from a lower level language (MAP) to a higher level language, namely decision tables, is developed in this chapter. In as much as the language of decision tables is a higher level language (61), it may be noted that conversion of FORTRAN programs to decision tables is itself a conversion from a lower level to a higher level language. Algorithms for such a conversion have been discussed in the previous chapters. Here we develop the algorithm of conversion from an assembly language program to decision tables.

In this chapter an approach which is different from the one discussed in Section 1.32 will be presented. Our objective is to get the logic behind assembly language programs through decision tables, rather than to translate them to machine language program. It must be noted that for this type of conversion also, one is likely to encounter some of the problems which are similar to these discussed before in Chapter I. With our approach, however, we shall be able to partly overcome some of the difficulties. It is felt that translation to an intermediate language is a desirable objective and D.T.s **fulfil** the requirements of an intermediate language (19). Decision Tables provide a <u>clear</u>

description of the logic involved in procedures. They form an orderly representation of information flow, from elementary decisions to final actions, and encourage consideration of different situations that arise in a problem. They are easy to visualise and update and hence serve as a useful document of programs,facilitating communication between people. D.T.s can be directly processed by the computer, unlike flowcharts which require coding before the machine can accept them. In addition, decision tables lend themselves to analytical treatment and it is possible to generate efficient computer programs from them.

With the new processors being developed, decision tables are likely to be directly acceptable to a processor. The use of D.T.s as excellent documentation and debugging aids has been high-lighted in the previous chapters. The decision table output will display the logic and it could be used as documentation, or with a certain amount of editing it may be used for translation as input to a processor which takes the D.T.s as input.

5.2. Conversion of Programs to Decision Tables: IBM 7044 FORTRAN VS. Assembly Language.

Before the algorithm for getting a decision table for a computer program (written in an assembly language) is presented, the following pertinent points should be noted.

1. Relative addressing is possible in most assembly languages .

Ex. TRA ALPHA + 3

2. The first executable statement in an assembly language may not necessarily be the first statement which will be executed. Assembly languages which have an END pseudo op, allow a label to be given in the variable field and when execution starts, control is transferred to this label.

Ex.

```
                 .
                 .
      BEGIN..............
                 .
                 .
                 .
                 .
                 .
      START..............
                 .
                 .
                 .
                 .
                END          START
```

3. Multiple entry points are allowed in subroutines written in assembly languages but not in FORTRAN.

4. Assembly language programs can have a 'transfer to' a symbol, which is not defined in the program; for example, it could be to an EXTERN label (35) or to a system generated EXTERN.

5. OP field can have a macro.

6. OP field can have an op which is defined or redefined in the program.

7. Pseudo ops (for 7044) like IFT, IFF allow conditional assembly of the instruction which follows these pseudo ops.

8. Certain transfer ops have actions associated with them. These actions take place before the actual transfer in some cases and after in certain other cases.

The implementation of the algorithm must take care of these additional factors and the implications of these points to a procedure (which may be similar to the one given in chapter II of this thesis) follows:

Because of relative and indirect addressing, Table A need not keep a column for the label (corresponding to statement number). Serial number is good enough. Because the OP field can contain an op which is defined or redefined, and also because of the point just mentioned before, table A can be formed only after a multiple scan.

To avoid table B from becoming too bulky, we could generate it from a list containing only the pertinent "transfer to" serial numbers. This also dictates a multiple scan for Phase 1.

Usually one is not interested in the logic for a very big assembly language program, one would like to get the logic for a small chunk of the program. Thus the user should be left with the choice of specifying the limits (STARTING/FINISHING) between which logic in the form of a decision table, is to be obtained. Any transfer outside these limits, then could be taken as an action. The processor could also take care of transfers to

statements which are not executable ones. This information, similar to,

```
            GO  TO   10
   10       FORMAT   (.......)
```

could help in pinpointing a few cases which load to illegal instruction traps.

With these observations, we pass on to algorithm.

5.3 Algorithm: MAP (IBM 7044) to Decision Table:

The given program is scanned from the first statement. While scanning, Table A is formed. All the statements or group of statements amounting to a compare, test, transfer or exit operation in the OP field are entered in this table (A). OP s defined and redefined, if belonging to any of the above mentioned categories, are also entered in Table A.

Table A consists of 7 columns. The first column contains the nature of an instruction. For a compare or test instruction, the condition being tested (Stub) is entered in column 1. This will need a certain amount of back tracking. After a compare or test instruction, a series of instructions which follow give the branching for such instructions. Transfers and CALL EXITs are entered as they appear. Column 2 has the serial number of the statement. Column 3 contains the type, like, compare, test, transfer, CALL Exit etc. etc. In columns 4,5 and 6, the 3 branches

```
*         STUDENT    HOMWORK-PICNIC   PROBLEM
BEGIN     CALL      S.READ(5,FMT1,HOMWRK,2)
          CAL       HOMWRK
          LAS       PENDNG
          TRA       *+2
          TRA       30.S
20.S      CAL       WETHR
          LAS       CLEAR
          TRA       *+2
          TRA       50.S
40.S      CALL      S.RITE(6,70FMT)
          CALL      EXIT
30.S      CALL      S.RITE(6,80FMT)
          TRA       20.S
50.S      CALL      S.RITE(6,80FMT)
          EXTERN    EXIT
          TRA       S.SJXT
FMT1      BCI       1,(2A6)
HOMWRK    BSS       1
WETHR     BSS       1
PENDNG    BCI       1,PNDNG
CLEAR     BCI       1,CLEAR
60FMT     BCI       2,GO ON PICNIC
70FMT     BCI       2,GO TO TEMPLE
80FMT     BCI       6,COMPLETE HOMEWORK MORNING
*            S.READ,S.RITE ARE FORTRAN COMPAT   I/O
          END       BEGIN
```

STARTING LIMIT 1.FINISHING LIMIT 27

FIG 5.1 <u>STUDENT HOMEWORK-PICNIC PROBLEM</u>

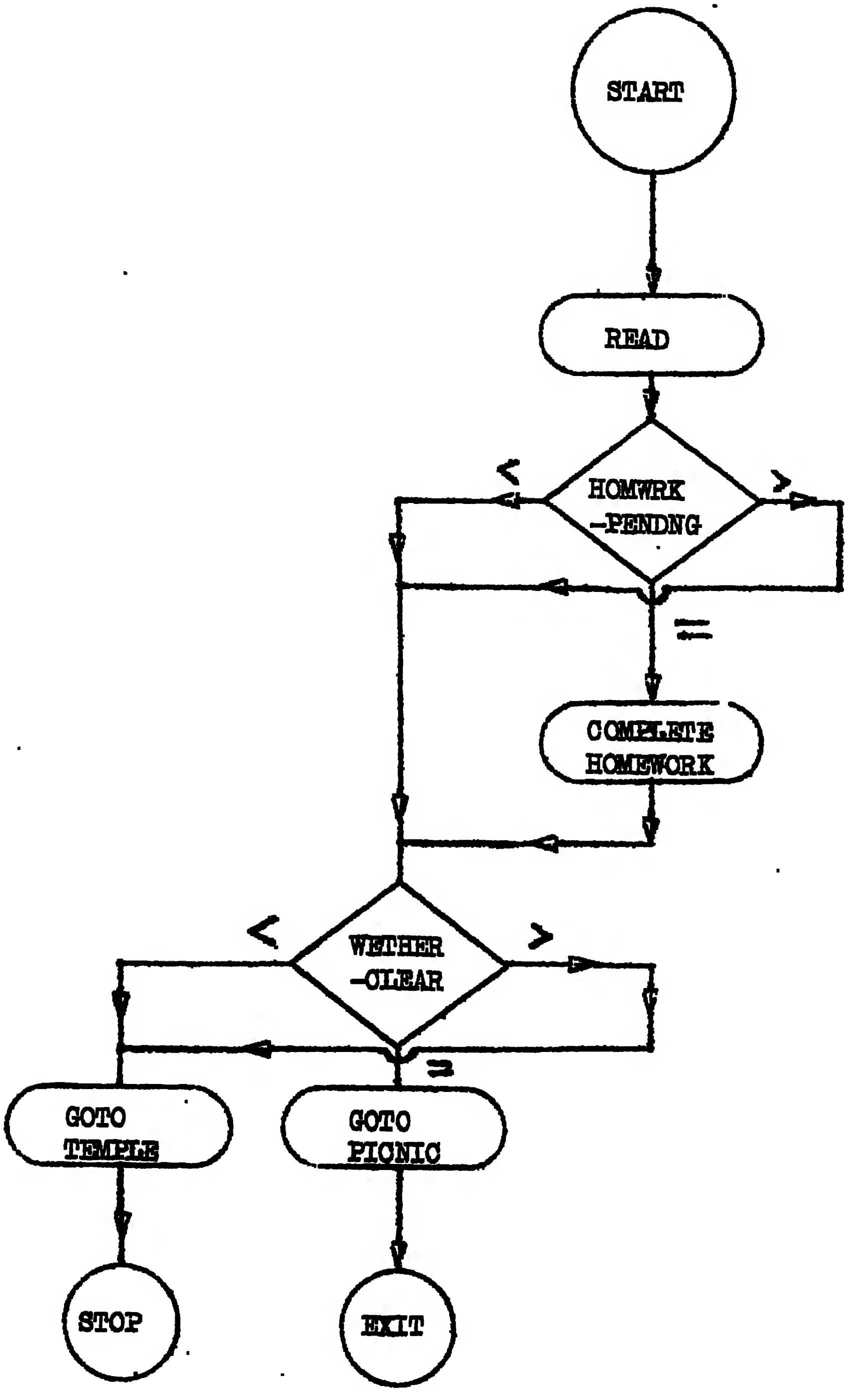


Fig. 5.2 Flowchart for Program of Fig. 5.1

for a compare instruction are entered. For test instructions, columns 4 and 6 contain the 'false' branch and column 5 corresponds to 'true' branch. As a transfer is an unconditional jump to a goal, the same serial number is entered in all the 3 columns (4,5 and 6). Here again to get the proper serial number for a goal, scanning of the label field would need a multiple scan. A CALL EXIT has no 'transfer to' statement serial number, and hence columns 4,5 and 6 are blanks. The seventh column is used as a temporary storage. Table A, corresponding to the sample program, written in the assembly language of IBM 7044 (MAP), Fig. 5.1, is given in Fig. 5.3.

From Table A a LIST containing the entries of columns, 4,5 and 6 is prepared, such that all the elements of this list are unique and arranged in ascending order, and none of the elements appear in column 2 of Table A. From this LIST another table B is generated. Table B has 3 columns. In the first column, the executable statement having a serial number corresponding to the elements of LIST, as it appears in the program is entered. The second column contains the serial number and the third column is a temporary storage. Table B, corresponding to the sample program (Fig. 5.1) is shown in Fig. 5.5a. The advantage of using LIST is that Table B gets reduced in size, with only the pertinent entries being retained. This would reduce the processing time. Using the above two tables the decision table is developed as follows:

1. In Table A, scan the row which has a serial number equal to or just greater than the STARTING LIMIT. In this case row serial number 4.

1	2	3	4	5	6	7
HOMWRK-PENDNG	4	COMPARE	7	13	7	
WETHR-CLEAR	8	COMPARE	11	15	11	4
CALL EXIT	12	STOP	--	--	--	8
TRA 20.S	14	TRANSFER	7	7	7	4
TRA S.SJXT	17	STOP	--	--	--	8

FIG 5.3 TABLE A

	1	2	3
20.S	CAL WETHR	7	4
40.S	CALL S.RITE..	11	8
30.S	CALL S.RITE..	13	4
50.S	CALL S.RITE..	15	8

FIG 5.5a TABLE B

7
11
13
15

FIG 5.4 LIST

HOMWRK-PENDNG	..	.NE.	.NE.	.EQ.	.EQ.
WETHR -CLEAR	..	.NE.	.EQ.	.NE.	.NE.
	..	11	15	13	13
	..	STOP	STOP	11	15
				STOP	STOP

FIG 5.5b DECISION TABLE

2. Enter condition in the condition stub of decision table (Fig. 5.5b). Go to step 3 if it is a test instruction (col. 3) otherwise see if the number in Br 1 is equal to Br 2 or Br 3. If yes, circle the appropriate entry. Enter .NE.0 in condition entry if Br 1 = Br 3, if Br 1 = Br 2, enter .LE.0. In all other cases it will be .LT.0. Go to step 4.

3. If it is a test instruction, entries in columns 4 and 6 will be identical and correspond to 'False' condition; circle the appropriate entry and enter F for false in condition entry. Then for Br2 we will have T for true.

4. Match the number in Br 1 against entries in column 2 of Table A. Go to the matching row. If match is not found go to step 6, otherwise, enter in column 7 of matching row the serial number of the statement from which the present statement was reached. Enter in condition stub of decision table the contents of column 1, unless it is for a CALL EXIT or a transfer.

5. Scan column 3 of matching row. If a compare or test instruction is found in this row, repeat step 2, otherwise go to step 8 if it is CALL EXIT (or TRA S.SJXT), else go to step 10 in case of transfer.

6. In this case no serial number is found to match with 7 in column 2 of Table A. In such a case go to Table B and match with column 2 of this table. Enter in TEMP of table B the serial number of statement in Table A, which led to it. Enter matching entry <u>serial number</u> as an action in the decision table.

7. Go to the next higher serial numbered statement in Table A. Enter in TEMP the serial number of statement in Table A which led to it (4 in this case). If the statement is STOP (or of similar nature) enter it as an action entry. If it is a condition, enter the condition and proceed as in step 2. In case of CALL EXIT or TRA S.SJXT, the next rule is to be found. For this go to step 8.

8. For the case of CALL EXIT, go to the row whose serial number is in column 7 (8 in this case).

9. Take the next uncircled branch of this row (Br 2 of row serial number 8 of Table A,which is 15 in this case) scan Table A, column 2 for a match of this number. If no match is found, go to Table B for a match and repeat step 7. In case all branches have been considered, that is, all branches are circled, go to the row whose serial number is in Temp, (in this case it is 4) after erasing the circles around Br 1, Br 2, Br 3 of the most recently tested condition.

10. The uncircled branch taken up is '7'. In row serial number 14 all branches are equal. It is thus an unconditional branch. It leads to table B which leads to serial number 8 of Table A. The branches of this condition are taken up one at a time (step 2) and the steps as before repeated.

11. When all branches are scanned, control is returned to serial number 4 which is the first condition tested in the program. As all branches for this condition have been exhausted, the

```
BEGIN   AXT     5,2
READ    CALL    S.READ(5,FMTIN,BUFFER,74)
        CALL    S.RITE(6,FMTOU,BUFFER,74)
        TSL     MATCH
        CALL    S.RITE(6,OUTPUT,INITAL,4)
        TIX     READ,2,1
        CALL    EXIT
        EXTERN  EXIT
MATCH   NOP     **
        CLA     =1
        STO     MAXNOP
        STO     UNPAIR
        CLA     FINAL
        ADD     INITAL
        PAX     ,1
        TXI     *+1,1,BUFFER-1
        SXA     NEXTCH,1
        LXA     FINAL,1
NEXTCH  CAL     **,1
        LAS     LFTPAR          IS IT (
        TRA     *+2                NO
        TRA     ADD1               YES
        LAS     RGTPAR        IS IT )
        TRA     *+2                NO
        TRA     SUBTR1             YES
        LAS     BLANK
        TRA     *+2
        TRA     EXITT              YES IT IS BLANK
CARYON  CLA     UNPAIR
        TZE     EXITT           IF IT IS MATCHING STRING EXIT
        ZAC                 LEST IT AMOUNTS TO OVERFLOW
        TIX     NEXTCH,1,1
        STZ     FOUND
        TRA*    MATCH
ADD1    CLA     =1
        ADD     MAXNOP
        STO     MAXNOP
        CLA     =1
        ADD     UNPAIR
        STO     UNPAIR
        TRA     CARYON
SUBTR1  CLA     UNPAIR
        SUB     =1
        STO     UNPAIR
        TRA     CARYON
EXITT   CLA     =100
        STO     FOUND
        TRA*    MATCH
LFTPAR  BCI     1,(
RGTPAR  BCI     1,)
```

Fig. 5.6 Continued on next page

```
BLANK  BCI    1,
UNPAIR BSS    1
FMTIN  BCI    2,(72A1,2I2)
FMTOU  BCI    3,(1H ,72A1,2X,2I6)
OUTPUT BCI    3,(1H ,4I12)
BUFFER BSS    72
INITAL BSS    1
FINAL  BSS    1
FOUND  BSS    1
MAXNOP BSS    1
       END    BEGIN
```

<u>Starting Limit 9</u>

<u>TABLE A</u>

TXI *+1	16	TRANSFER	17	17	17	-
ACC=LFTPAR	20	COMPARE	23	35	23	B 32
ACC=RGTPAR	23	COMPARE	26	42	26	20
ACC=BLANK	26	COMPARE	29	46	29	23
UNPAIR=0?	30	TEST	31	46	31	26 45
XR1.GE.1?	32	TEST	33	19	33	30
TRA* MATCH	34	RETURN	-	-	-	32
TRA CARYON	41	TRANSFER	29	29	29	20
TRA CARYON	45	TRANSFER	29	29	29	23
TRA* MATCH	48	RETURN	-	-	-	30 26 30

<u>LIST</u>

17	33
19	35
29	42
31	46

(Fig. 5.6 continued on next page.)

TABLE B

	SXA	NEXTCH,1	17	-
NEXTCH	CAL	**,1	19	32
CARYON	CLA	UNPAIR	29	26
	ZAC		31	30
	STZ	FOUND	33	32
ADD1	CLA	=1	35	20
SUBTR1	CLA	UNPAIR	42	23
EXITT	CLA	=100	46	30 26

DECISION TABLE

RULE

Y

ITION	1	2	3	4	5	6	7	8	9	10
LFTPAR	N	N	N	N	N	N	N	Y	Y	Y
RGTPAR	N	N	N	N	Y	Y	Y			
BLANK	N	N	N	Y						
IR=0	N	N	Y		N	N	Y	N	N	Y
GT.1	N	Y			N	Y		N	Y	
ONS										
	29	29	29	46	29	29	29	35	35	35
	31	31	46	RETURN	31	31	46	31	31	46
	33	19	RETURN		33	19	RETURN	33	19	RETURN
	RETURN	SELF			RETURN	SELF		RETURN	SELF	

The resulting decision table is shown in Fig. 5.5b. Another example of a MAP program and the resulting decision table are shown in Fig. 5.6.

5.4 Assembly Language to Decision Tables:

Conversion of a MAP program to decision tables was described in the last section. A similar procedure can be used for any assembly language. In fact the procedure outlined works for almost any language, once tables A and B are formed. Thus the only major difference lies in the procedure by which the two tables are formed. For the success of the procedure, classification of operation codes is important. One such classification of op codes (36) for MAP is given in Fig. 5.7. It will be noted that we are keeping the unconditional transfer TRA in a category which is different from the one having op codes which have actions associated with them. An implementation must take care of this fact. Similarly pseudo operation codes like IFF and IFT which allow conditional assembly of the instructions which follow must also be taken care of. To illustrate, how the procedure given before in the last section can be used with very minor changes, another assembly language will be considered.

Student Home Work - Picnic problem coded in the assembly language of IBM 1401 (AUTOCODER) is given in Fig. 5.8 and the corresponding LIST, Tables A and B in Fig. 5.9. To bring out the similarity, the variable names and labels used are almost similar

TEST	COMPARE	TRANSFER	TRANSFER with Action	SUBROUTINE TYPE	STOP RETURN
DCT	CAS	TRA	TRP	TSL	TRA*
ETTA	CCS		TRT	CALL	CALLEXIT
IOT	LAS		TSX	XEC	TRA S.SJXT
LBT			TXI		
MIT					
PBT					
PLT					
SWT					
TCOA					
TDOA					
TEFA					
TIX					
TMI					
TNX					
TOV					
TPL					
TRCA					
TXH					
TXL					
TZE					

Pseudo ops

IFT

IFF

Fig. 5.7 Classification of op. codes.

```
AUTOCODER RUN   THRU OUTPUT
          JOB   AUTOCODER TRIAL
          CTL   4111 PPP    S
*         STUDENT HOMEWORK-PICNIC PROBLEM
BEGIN     CS    331             CLEAR PRINT AREA
          CS
          SW    5,10            PUT WORDMARKS,CARD IN. AREA
          R                     READ A CARD
          MLC   001,HOMWRK
          MLC   006,WETHR
          C     HOMWRK,PENDING SEE IF HOMEWORK IS PENDING
          BE    S30
S20       C     WETHR,CLEAR     CHECK WETHER CONDITIONS
          BF    S50
S40       MCW   FMT70,200       MOVE CHARACTER TO WORD
          W     HALT
S30       MCW   FMT80,200
          W                     COMPLETED HOMEWORK
          CS    350,S20
          B     S20
S50       MCW   FMT60,200
          W
HALT      H
HOMWRK    DCW   '     '         DEFINE CONSTANT WITH WORD MARK
WETHR     DCW   '     '
PENDING   DCW   'PNDNG'
CLEAR     DCW   'CLEAR'
FMT60     DCW   'GO ON PICNIC'
FMT70     DCW   'GO TO TEMPLE'
FMT80     DCW   'COMPLETE HOMEWORK MORNING'
          END   BEGIN
```

FIG.5.8 STUDENT HOMEWORK PROBLEM IN AUTOCODER

	1	2	3	4	5	6	7	
HOMWRK-PENDING		8	COMPARE	10	14	10		
WETHR-CLEAR		10	COMPARE	12	18	12	8	17
W	HALT	13	BRANCH	20	20	20	10	
B		17	BRANCH	10	10	10	8	
H		20	STOP	-	-	-	13	10

TABLE A

	1	2	3
MCW	FMT70,200	12	10
MCW	FMT80,20	14	8
MCW	FMT60,200	18	10

TABLE B

12
14
18

LIST

HOMWRK-PNDNG	..	.NE.0	.NE.0	.EQ.0	.EQ.0
WETHR -CLEAR	..	.NE.0	.EQ.0	.NE.0	.EQ.0
....					
		12	18	14	14
		STOP	STOP	12	18
				STOP	STOP

DECISION TABLE

FIG.5.9 TABLE A,TABLE B,LIST ETC. ETC. FOR FIG.5.8

to those used in the MAP program given in Fig. 5.1. Thus it will be seen that the main difference in the procedure, for getting a decision table corresponding to a program written in an assembly language, lies in the way we generate Tables A and B: having got the structure of the program in the form of Tables A and B, the rest of it is tracing the various paths of the tree and is bound to be similar.

5.5 Additional Features of Assembly Languages:

In section 5.2 a number of differences between programs written in FORTRAN and assembly languages were discussed. A few other factors have also to be kept in mind.

1. Assembly languages allow same lable to be used in different portions of the program without getting the MULTIPLY DEFINED error by separating the different sections by pseudo ops like QUAL and ENDQ. Thus any label used within a qualification section becomes a local variable. When a local label LL within a qualification section say XX is to be referred to from outside it is referred as XX$LL. Care should be taken for such labels while generating Table A.

2. FORTRAN does not have any instruction like XEC (Execute).

There are program segments like B (Fig. 5.10) which

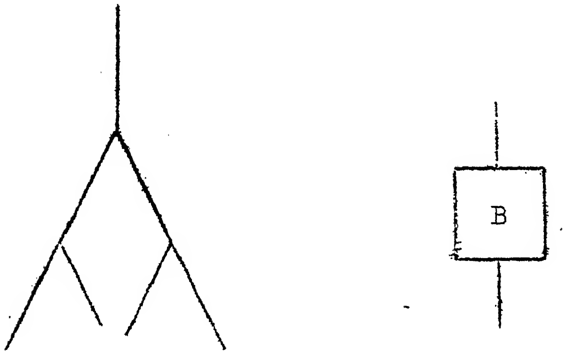


Fig. 5.10 Apparently Redundant Program Segment.

appear to be redundant (Section 4.1) at first glance but are not: different instructions of such a program segment are accessed by XEC instructions. An example from SCAN routine of IBM 7044 FORTRAN-IV compiler is given below (Fig. 5.11)

```
        XEC     TRNTAB, 4
         .
         .
        TRA     ERCLIO      77
         .
        TRA     ERCLIO      74
         .
        TRA     ERCLIO      73
         .
        TRA     ERCLIO      72
         .
         .
         .
        TRA
         .
         .
TRNTAB  TRA     ERCLIO      00
```

77 Octol statements having TRA s in the op. field

Fig.5.11 Portion of IBM 7044 FORTRAN-IV Compiler - 'SCAN'.

In such cases the 'transfer to' depends upon the contents of index register 4. The author feels that it is difficult and expensive (in terms of computer time) to decode the logic in such dynamically changing program segments.

5.6 Conclusions:

An algorithm which would convert an assembly language program to decision tables has been presented.

Programs which generate decision tables corresponding to syntactically correct assembly language programs can be used for program documentation, as diagonastic aids for semantic or logical checking of programs, and also as an intermediate language in the automatic translation of assembly language programs from one computer to another. With a certain amount of pre and post editing, this approach would help in the decompiler problem.

CHAPTER VI

CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

In the previous chapters, a solution to two important, though seemingly different problems facing computer scientists, namely detection of logical errors in programs and of converting programs written for one computer to be run on another, has been discussed. Use of decision tables as an intermediate step in solving both these problems has been suggested. Algorithms for converting computer programs to decision tables have been presented. As programs for converting decision tables to machine language are available, decision tables, generated from computer programs, would be a good intermediate language in converting programs written for one computer to those of another or for program documentation (as a replacement of flow charts) and also as diagonastic aids for semantic or logical checking of programs.

It must be emphasised at the outset, that the endeavour has been to test the feasability of automated conversion of programs to decision tables. This is an area in which the potentiality of decision tables had not been explored so far. The present venture is a beginning in this direction, and hence by no means 'the last word'.

A number of suggestions for improving the implementation of the algorithm for converting FORTRAN programs to Decision Tables

have been given earlier in section 3.8. It would be worthwhile modifying Phase 1 (Section 3.2) so that we get 'S' matrix and then could get an idea about spurious blocks, endless loops, if any, and about relatively independent sub-sections of the sample program. The latter, it is felt, would be very useful to debug and document very lengthy programs. For segmenting the sample program, Meakawa's (53) procedures, based upon the list technique suggested by Krider (50) could be useful.

It has been shown that the philosophy of the algorithm works for languages other than FORTRAN also, though the details of how the algorithm would be implemented depend on the language being considered. Algorithm for converting MAP program to decision tables (Chapter V) is very similar to that given in Chapter II (FORTRAN to D.T.s). Because of this, an altogether different approach, similar to one given by Sherman (85) for flowcharting, may be to develop a general processor which would accept all languages. The technique would be to describe the syntax of the control statements of the language used for the sample program, in a meta language, and feed the sample deck to such a processor to get the decision table.

For programs which modify themselves, the algorithms presented in this thesis will not work. Before one attempts to get decision tables for self modifying programs, one shall have to answer whether a self modifying decision table could be defined,

and if so, whether it will aid human thinking in program debugging.

With the techniques discussed in this thesis it is possible to pointout the mistakes in logic. However, an ideal solution would be to pointout the particular statement or a set of such statements which, when corrected would result in correcting the mistakes in logic. In the absence of this ideal solution, the next best solution would be to narrow down the region wherein lie the mistakes in logic.

A different approach to tackle the same problem would be to go in for a very detailed condition stub analysis. It might be worthwhile, from the point of view of further pinpointing the bugs in a program.

REFERENCES

1. ACM, (1968) "Professional Development Seminar; Decision Tables for Computer System Design and Programming".

2. Anderson, H.E., (1965) "Automated Plotting of Flowcharts on a Small Computer", Scandia Corporation, Albuquerque.

3. Armerding, G.W., (1962), "FORTAB: A Decision Table Language for Scientific Computing Applications", Rand Corporation, RM-3366/PR.

4. Bar-hillel, Yehoshua, (1960) "The Present Status of Automatic Translation of Languages", Advances in Computers, Vol. 1, pp 91-163.

5. Benjamin, R.I., (1965) "The Spectra 70/45 Emulator for the RCA 301", Com. ACM, Vol. 8, No. 12, pp. 748-752.

6. Boerdam, Wim, (1967) "Decision Tables in System Design", Proc. Spring Joint Computer Conference.

7. Cantrell, H.N., King, J., and King, F.E.H., (1961) "Logic Structure Tables", Com. ACM, Vol. 4, p. 272.

8. Chapin, Ned , (1967) "Parsing of Decision Tables", Com. ACM, 10, No. 8, pp. 507-511.

9. Codasyl-Jug., (1962) "Proc. of the Decision Table Symposium", Sponsored by the Codasyl and the Jug of ACM in New York.

10. Cress, Paul, Dirksen, Paul and Grahm, Wesley, Jr., (1968) "FORTRAN-IV with Watfor", Printice Hall, Inc. (Book), p. 143.

11. Dellart, George T., (1965) "A use of Macros in Translation of Symbolic Assembly Language of One Computer to Another", Com. ACM., Vol. 8, No. 12, pp. 742-748.

12. Dixon, Paul , (1964) "Decision Tables and Their Applications", Computers and Automation, April.

13. Evans, Orren Y., (1961) "Advanced Analysis Method for Integrated Electronic Data Processing", IBM General Information Manual, F20-8047.

14. Egler, J.F., (1963) "A Procedure for Converting Logic Table Conditions into an Efficient Sequence of Test Instructions", Com. ACM, Vol. 6, p. 510.

15. Fergusan, Earl H. and Berner, Elizabeth , (1963) "Debugging Systems at the Source Language Level", Com. ACM., Vol. 6, No. 8, pp. 430-432.

16. Fisher, D.H., (1966) "Data, Documentation and Decision Tables", Com. ACM., Vol. 9, No. 1, pp. 26-31.

17. Fisher, F. Peter and Swindle, George F. (1964) "Computer Programming Systems", Holt, Rinehart and Winston (Book).

18. Elores, Ivan (1966) "Computer Programming", Printice Hall (Book).

19. Gains, R. Stockton (1965) "On the Translation of Machine Language Programs", Com. ACM., Vol. 8, No. 12, pp. 736-741.

20. Ganpati, S., (1969) "Information Theory Applied to Decision Tables", M. Tech. Thesis, Electrical Engineering Department, IIT-Kanpur.

21. General Electric (1965) "GE-200 Series GECOM-II COBOL Compatible Operations Manual", Program No. CD225H1.005.

22. Goetyz, Martin A., (1967) "Recent Developments in Automated Program Documentation", Applied Data Research, Inc., Princeton, New Jersey.

23. Grindley, C.B.B.,(1966) "Systematics - a non-programming Language for Designing and Specifying Systems for Computers", Comp. Jour., Vol. 9, p. 124.

24. Gunn, J.H.,(1966) "Problems in Program Interchangeability", Proc. Symposium organised by International Computation Centre, Rome, March.

25. Gupta, Virendra, (1967) "Computer Flow Charting", M. Tech. Thesis., Electrical Engineering Department, IIT-Kanpur.

26. Gupta, Virendra and Rajaraman, V., (1969) "Checking of Program Logic with Decision Tables" (Communicated).

27. Haibt, Lois M., (1959) "A Program to Draw Multilevel Flowcharts", Proc. Western Joint Comput. Conf. pp. 131-137.

28. Halpern, Mark I., (1963) "Machine Independence: Its Technology and Economics", Com. ACM., Vol. 8, No. 12, pp. 782-785.

29. Halpern, Mark I., (1968) "Towards a General Processor for Programming Languages", Com. ACM., Vol. 11, No. 1, pp. 15-25.

30. Heart, Jane and Reiner, David , (1959) "Flowchart Analysis Program", Lincoln Lab., Lexington, Mass.

31. Holstien, David,(1962) "Decision Tables, a Technique for Minimising Routine, Repetitive Design", Machine Design, Vol. 34, August 2, pp.76-79.

32. IBM "Flowcharting Techniques", Form C20-8152.

33. IBM "Decision Tables: A System Analysis and Documentation Technique", Form F20-8102-0

34. IBM (1964) "Decision Logic Translation", IBM Application Program H20-0063.

35. IBM (1965) "Macro Assembly Program (MAP) Language", Form C28-6335-2.

36. IBM (1965) "7040 and 7044 Data Processing Systems - Student Text", Form C22-6732-2.

37. IBM (1965) "Debugging Facilities", Form C28-6803-1.

38. IBM (1966) "Programmers Guide", File No. 7040-36 Form C28-6318-7.

39. Irons, E.T.,(1965) "A Rapid Turn Arround Multiprogramming System", Com. ACM., Vol. 8, No. 3, pp. 152-157.

40. Jacoby, K. and Layton, H., (1961) "Automation of Program Debugging", Philco Corpn. PC No. 859.

41. Karp, Richard M., (1960) "A Note on the Application of Graph Theory to Digital Computer Programming", Information & Control, Vol. 3, No. 2, June, pp. 179-189.

42. Katz, Jenold J. and Foder, Jerry A., (1963) "The Structure of a Semantic Theory", Language, Vol. 39, No. 2, Part 1, April-June, pp. 170-210.

43. Kavanaugh, T.F.,(1961) "TABSOL, the Language of Decision Making", Comp. Auto.,Vol. 10, No. 9.

44. Kelkar, S.P.,(1968) "A Package Program for Survey Data Processing with Digital Computer", Masters Thesis EE-4-1968, IIT-Kanpur.

45. King, P.J.H.,(1967) "Decision Tables" Computer Journal, Vol. 10, No. 2, pp. 135-142.

46. King, P.J.H., (1966) "Conversion of Decision Tables to Computer Programs by Rule Mask Techniques", Com. ACM., Vol. 9, No. 11, pp. 796-801.

47. Krik, H.W., (1965) "Use of Decision Tables in Computer Programming", Com. ACM., Vol. 8, No. 1.

48. Koulagina, Olga S., (1962) "The Use of Computers in Research in Machine Translation", Proc. IFIP Congress 62, Munich.

49. Kramer and Kirk, (1966) "Decision Table Technique in Computer Control", IEEE Trans. Power and Apparatus, May, pp. 495-498.

50. Krider, Lee, (1964) "A Flow Analysis Algorithm", Jour. ACM., Vol. 11, No. 4, pp. 429-436.

51. Knuth, Donald E., (1963) "Computer Drawn Flowcharts", Com. ACM., Vol. 6, No. 9, pp. 555-563.

52. Larsen, R.P., (1966) "Data Filtering Applied to Information Storage and Retrieval Applications" Com. ACM., Vol. 9, p. 785.

53. Maekawa, Mamoru, (1968) "Automatic Flowcharting", Information Processing in Japan, Vol. 8, pp. 72-81 (Original in Japanese Joho Shori, Vol. 9, No.3, pp. 129-136.)

54. Marimont, Rosalind B., (1959) "A New Method of Checking the Consistency of Precedence Matrices", Jour. ACM., Vol. 6, pp. 164-171.

55. McCormack, M.A., Schansman. T.T. and Womach, K.K., (1965) "1401 Compatibility Feature on the IBM System/360, Model 30 , Com. ACM, Vol. 8, No. 12, pp. 373-376.

56. McDanial, Herman, (1968) "An Introduction to Decision Logic Tables", John Wiley & Sons. (Book).

57. Miller, George A., (1956) "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", Psychological Review, Vol. 63, No. 2, pp. 81-97.

58. Miller, Joan C. and Maloney, Clifford J., (1963) "A Method for Systematic Error Analysis of Digital Computer Programs", Com. ACM., Vol. 6, No.2, pp. 58-72.

59. Minsky, Marvan L., (1967) "Computation: Finite and Infinite State Machines", Printice Hall, Inc., (Book), pp. 23-26.

60. Montalbano, M., (1962) "Tables, Flowcharts and Program Logic", IBM Systems Journal, pp. 51-63.

61. Muthukrishnan, C.R. (1969) "Analysis and Conversion of Decision Tables to Computer Programs", Doctoral Thesis, Electrical Engineering Department, IIT-Kanpur, India.

62. Nagao, Makoto, (1965) "An Approach to the General Theory of Natural Languages", International Conference on Computational Linguistics, Tokyo.

63. O'Brien, F. and Beckwith, R.C., (1968) "A Technique for Computer Flowchart Generation", Computer Journal, Vol. 11, No. 2.

64. Oerter, G.W., (1968) "A New Implementation of Decision Tables for a Process Central Language", IEEF Trans. on Industrial Electronics, Control and Instrumentation.

65. Opler, A., et. al. (1962) "Automatic Translation of Programs from one Computer to Another" Information Processing, pp. 550-553.

66. Oslen, Thomas M., (1965) "Philco/IBM Translation at Problem Oriented, Symbolic and Binary Levels", Com. ACM, Vol. 8, No. 12, pp.762-768.

67. Pierce, John R, (1968) "Man, Machines and Languages" IEEE Spectrum, July.

68. Pollack, S.L., (1963) "Analysis of Decision Rules in Decision Tables" RM3669-PQ, Rand Corpn., Santa Monica, Calif.

69. Pollack, S.L., (1965) "Conversion of Limited Entry Decision Tables to Computer Programs", Com. ACM, Vol. 8, No. 11.

70. Press, Laurence I., (1965) "Conversion of Decision Tables to Computer Programs", Com. ACM, Vol. 8, No. 6.

71. Prosser, R.T., (1959) "Application of Boolean Matrices to the Analysis of Flow Diagrams" Proc. Eastern Joint Comput. Conf., No. 16, p. 133.

72. Ramana Rao, K.V., (1969) "An Approach to a General Programming Language Processor Through Decision Tables", Masters Thesis, Department of Electrical Engineering, IIT-Kanpur.

73. Reinwald, Lewis T. and Soland, Richard M., (1966) "Conversion of Limited Entry Decision Tables to Optimal Computer Programs" Pt. I, Jour. ACM, Vol. 13, pp. 339-358.

74. Reinwald, Lewis T. and Soland, Richard M., (1967) "Conversion of Limited Entry Decision Tables to Optimal Computer Programs" Pt. II, Jour. ACM, Vol. 14, No. 4, October.

75. Sahasrabudha, H.V.,(1967) "Problems of Deassembly", Department of Electrical Engineering, IIT-Kanpur (Private Memo).

76. Sammat, Jean E.,(1967) "Fundamental Concepts of Programming Languages", Computers and Automation, February.

77. Sassaman, William A., (1966) "A Computer Program to Translate Machine Language Into FORTRAN", Proc. Spring Joint Comput. Conf., Vol. 28, pp. 235 - 239.

78. Satterthwait, Arnold C. (1966) "Programming Languages for Computational Linguistics", Advances in Computers, Vol. 7, p. 209.

79. Satyavan, A.R.,(1969) "Computerised Production Planning" M. Tech. Thesis, Electrical Engineering Department, IIT-Kanpur.

80. Scott A.E.,(1958) "Automatic Preparation of Flowchart Listings", Jour. ACM., Vol. 5, No. 1, pp. 57-66.

81. Senko, M.E.,(1960) "A Control System for Logical Block Diagonosis with Data Loading", Com. ACM, Vol. 3, No. 4, pp. 236-240.

82. Seshagari N., (1967) "Relay Tree Network Decomposition of Decision Tables" (Letter) Proc. IEEE, Vol. 55, No. 9, p. 1648.

83. Seshu, Sundram and Reed, Myrill B., (1961) "Linear Graphs and Electrical Networks", Addison-Wesley (Book) p. 61.

84. Sherman, P.M., "Fortrace, a 7090 Computer Program for Flowcharting FORTRAN Programs", Bell Telephone Labs. (Private Memo.).

85. Sherman, P.M.,(1966) "Flowtrace, A Computer Program for Flowcharting Programs", Com. ACM, Vol. 9, No. 12, pp. 845-854.

86. Tucker, S.G.,(1965) "Emulation of Large Systems", Com. ACM, Vol. 8, No. 12, pp. 753-761.

87. USERS Guide for LOGITRAN (1969) Prepared by Gupta, Virendra, Computer Centre, IIT-Kanpur.

88. Veinott, Grill G.,(1966) "Programming Decision Tables in FORTRAN, COBOL or ALGOL "Com. ACM, Vol. 9, No. 1, pp. 31-35.

89. Veinott, Grill G.,(1966) "More on Programming Decision Tables", Com. ACM. Vol. 9, No. 7, p. 485.

90. Voorhees, Edward A.,(1958) "Algebraic Formulation of Flow Diagrams", Com. ACM, Vol. 1, No. 1, pp. 4-8.

91. Warshall, Stephon,(1962) "A Theorem on Boolean Matrices", Jour. ACM,Vol. 9, No. 1, pp. 11-12.

92. Wilde, Danial U., (1966) "Program Analysis by Digital Computer" Doctoral Thesis, Department of Electrical Engineering, M.I.T., MASS.

APPENDIX I

PROBLEM OF SEMANTICS

IN

MACHINE TRANSLATION OF LANGUAGES

From the present state of the art in machine translation of languages, it is felt that the most difficult problem being faced is the multiple meaning of words and phrases, which is intrinsically a problem of semantic theory. A few examples will make this point clear.

Example 1. (4) (Bar-Hillel, Appendix III)

The box was in the pen.

The linguistic context from which this sentence is taken is, say the following:

Little John was looking for his toy box. Finally he found it. The box was in the pen. John was very happy.

Assuming, for simplicity that pen in English has only the following two meanings: (i) a certain writing device (ii) an enclosure where small children can play; in the current state of the art, no existing or imaginable program will enable a computer to determine, that the word pen in the given sentence, within the given context, has the second of the above meanings, whereas every reader with a sufficient knowledge of English will do this "automatically".

Example 2.

Tom and Dick went to the store and the movies respectively.

Helpern (28) writes "The key to the difficulty, is that "respectively" is essentially a model, not a notational element; like a pseudo op in an assembly language, it is not so much input to the translation process as it is an adjustment of the translator, forcing it to operate in a special mode".

Example 3. (67):

TIME FLIES LIKE AN ARROW

We instantly know the meaning but a computer may well conclude that there are timeflies who like an arrow, or that someone is being instructed to time flies in the same manner as an arrow would time flies. There are other grammatically sensible interpretations of this sentence as well .

The "Concise Oxford Dictionary of Current English" (Fowler and Fowler, 1942) gives as many as 4 different ways in which PEN can be used. In the first example given above the difficulty was of contextual nature. Whereas in the second example given above, there, is a certain amount of ambiguity present. At present the semantic theory of languages has no powerful method of itself, which is able to account for the exact determination of the meaning of sentences.

APPENDIX II

FILE 99 (TAPE 99)

There are several occasions in the handling of data when reformating is desirable. Instructions using FORTRAN logical unit 99 provides this facility. The FORTRAN Read, Write Commands when performed on file 99, cause the transmission of data to occur between memory locations.

The READ and WRITE statements with logical unit 99 are of the form

READ (99, Format) List

and WRITE (99, Format) List

which are referred to us READ 99 and WRITE 99 statements. Logical units 0 to 4 are attached to tape units. So, whenever read and write statements refer to these units, the data is read and written respectively from the corresponding tape unit. But the logical unit 99 has been created without any physical input-output (I/O) device attached to it. Thus a WRITE 99 instruction merely transfers the contents of the specified lists, under the given format into an area in memory, where as READ 99 statement uses the contents of this area of memory to read data into the specified variables under the given format. Consecutive READ 99 statements make use of the same area of memory to form different lists under several format specifications. Since reading from memory is nondestructive, rereading the same area of memory is possible. This eliminates the

concern for the user about REWINDS, BACKSPACING etc. etc.

For this READ, WRITE 99 feature, a file FIL99 has been created, with no physical units attached to the file. (Deck F99, IIT/K IBSYS may be referred). IOCS assigns buffers to this file at the time of loading. There is no need to <u>open</u> the file for executing READ (99, n) List, since the reading is not done from any physical unit, but the list is taken from the buffer created in the RWD routine. Likewise to execute WRITE (99,n) List, <u>opening</u> the file is necessary only once in the beginning. Changes have been made in the RWD routine of the FORTRAN IV subroutine library. A buffer

```
BUFF    BSS    22
```

has been created. The execution takes the same path as in the case of usual I/O statements, but at each stage (opening for READ, opening for WRITE and actual transmission of data), check is made to find out whether the file under consideration is FIL99. If so, the path taken is slightly different.

APPENDIX III

LISTING OF LOGITRAN

A complete listing* of LOGITRAN is given in next and subsequent pages; on page 184 is given the listing of subroutine PATERN, referred to on page 55 of chapter III.

*Because of the limitation of size of a page, only contents of columns 1 to 64 of a computer card can fit in. A program was written to punch contents of columns 65-72 as a continuation card. In this process a few words, here and there, have got split. Such cases are indicated by a "/" in column6.

```
$IBFTC FORTAB  NODRNT
C
C
C             .------------------------------------.
C             I                                    I
C             I            LOGITRAN                I
C             I     PROGRAM FOR CONVERTING         I
C             I              LOGIC                 I
C             I              OF A                  I
C             I        COMPUTER PROGRAM            I
C             I              TO A                  I
C             I         DECISION TABLE             I
C             I                                    I
C             I                                    I
C             I   WRITTEN BY                       I
C             I        VIRENDRA  GUPTA             I
C             I         COMPUTER CENTRE,           I
C             I          I.I.T./KANPUR             I
C             I                                    I
C             I                NOVEMBER,1969       I
C             I                                    I
C             .------------------------------------.
C
      INTEGER BCDTAB(400),RE,RETURN,CA,HEADER(100),SUBTAB,ELSE
      INTEGER RENTER(150),SELF,SLASH,MASTER(150)
      DATA SELF/4HSELF/,SLASH/1H//
      INTEGER NUMB(32),ENDING,T
      INTEGER COMMA,PLUS,FILLED,RETU
      INTEGER  ALPBET(26),ALNUM(36),ANDD,BUF78,BUF79,BUF712,BUF7
     /10,
     1 BLANK,BLANKS,BEGIN,BUFFER(600),BRNCH(150,3),BRNCH1,BRNCH2
     /,BRNCH3
     2,C,COL6,COL1,COL16,CONDSN(40,7),COLS(80),CALL,ENTRY(150,7)
     /,
     3ENTRYS(7),EXENT(40,40),EQZ,EXCUTD(150,3),EXINT,D,DO,DELTRS
     /(5)
C     IF YOU CHANGE DIMENSIONS OF EXENT, LOOKUP ..CMPL.. ROUTINE
     4,DLIMIT,GTZ,GEZ,GOTO4,GOTO,GOTOLT,GO, FINAL,FOLLOW,FVALUE,
     /FARMAT(5
     4),IFLEFT,INDEX,LTZ,LEZ,LIST(40,40),LOGOPR,OBTAIN,OPRATR,OP
     /RTRS(8)
     5 ,PREVCH,PREVNO(150),NUMBER(10),N(400),NEZ,NEED,RULE,RETFR
     /M,ROWT
     6A1,ROWTA2,RESET,RANGE,STOP4,STARS,STMNT(150),STOP,
C     IF YOU CHANGE DIMENSIONS OF BCDTAB, CHANGE THOSE OF ..N..
C    /ALSO
     7ST,STEP,STMNNO,TABLE,TERM1,TERM2,TYPE
      INTEGER PERIOD,EQUALS,ORR,FOUND,LONGEX(20),UPLIMT,ORDER,FU
     /LWRD,REM
     1AIN,DASH,PREVCD,DECKEN,DEKEND,RIGHTP,GOAL,EXITT
```

```
 LOGICAL SUBDT,LOGIFO
 DATA INARTH,PLUS,RETURN,RETU/5HI/O.A,2H++,6HRETURN,4HRETU/
 DAT/    ALPBET                  ,          BLANK,BLANKS,CALL/1HA,1HB,1
/HC,1HD,1
2HE,1HF,1HG,1HH,1HI,1HJ,1HK,1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS
/,1HT,1HU
3,1HV,1HW,1HX,1HY,1HZ,          1H ,5H       ,4HCALL/,  DELTRS
4 ,DLKEND,DASH,DO,  NUMBER                        /1H,,1H.,1H(,1H),
/1H=,6HDE
5KEND ,5H   -   ,2HDO,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,1H0
//,
5 GO/2HGO/,GOTOLT/5HGOTO(
6/,IF,NONEED, OPRTRS              /2HIF,6HNONEED,5H.LT.0,5H.LE.
/0,5H.EQ.
70,5H.GE.0,5H.GT.0,5H.NE.0,5H.AND.,4H.OR./,STOP4,
/ ST/4HST
8OP,                   2HST/,4INUS/1H-/,FARMAT(1),FARMAT(3),F
/ARMAT(5)
9/1H(,4HA6,A,1H)/
 DATA  GOTO4,IFLEFT/4HGOTO,3HIF(/,EXITT/4HEXIT/
 DATA CONDTN/6HCONDTN/,RE,CA/2HRE,2HCA/
 DATA AS/2HAS/,NOT/5H.NOT./
 EQUIVALENCE  (COMMA,DELTRS(1)),(PERIOD,DELTRS(2)),(LEFTP,D
/ELTRS(3)
1),(RIGHTP,DELTRS(4)),(EQUALS,DELTRS(5)),(C,ALPBET(3)),(D,A
/LPBET(4)
3),(LTZ,OPRTRS(1)),(LEZ,OPRTRS(2)),(EQZ,OPRTRS(3)),(GEZ,OPR
/TRS(4)),
4(GTZ,OPRTRS(5)),(NEZ,OPRTRS(6)),(ANDD,OPRTRS(7)),(ORR,OPRT
/RS(8))
 EQUIVALENCE  (T,ALPBET(20)),(LOOP,UPLIMT)
 EQUIVALENCE (ROWTA1,NCRD),(ROWTA2,NOTAB2)
 COMMON//BRNCH,CONDSN,ENTRY,EXENT,EXCUTD,NCRD,N,PREVNO,STMN
/T,LIST,
1 MIN1,LF,LFF,IRULES,RULE,RETFRM
3 ,HEADER,SUBTAB,NOSTRO,ENTRYS
 COMMON /RULES/ NUMB,ELSE, LOGDT,NOSUBT
 COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
 COMMON/DETAB/BEGIN,GOTO,STOP,STARS
 COMMON/LOOKUP/BCDTAB
```

```
C----------------  -------- ------------------  ---------- ---
C
C
C                      LIST  AND  APPEARENCE OF  COMMON  BLOCKS
C
C
C----------------  -------- ------------------  -------------
C
C                      APRIL 18,1969
C      8               20          30
C      NAME            LENGTH     APPEARANCE    LIST       REMARKS
C
C----------------  -------- ------------------  -------------
C
C      //              5870       FORTAB  MERGE  PARSE  REDUCE
C     COMPUTE THE NEW LENGTH OF // AND CHANGE IN PARSE AND CO.
C      RULES           35         FORTAB  MERGE  BLOCK
C      SOURCE          604        FORTAB  REPLAC  REPCOL  SEQUNC
C                                 GETCON  SQUEZE  GETENT  PUTBRN
C                                 SEARCH  NODLMT  DTENT
C      DETAB           4          FORTAB  MERGE  DTENT  BLOCK
C      LOOKUP          200        FORTAB  DTENT
C      OPS             6          BLOCK
C
C---------------- - -------- ------------------- -------------
C
C
C---------------- --------- ------------------- -------------
C
C*****************************************************************
C
C                      FEBRURARY  1969
C
C     DIMENSIONS OF THE FOLLOWING VARIABLES WERE DECIDED MORE
C     OR LESS ON AN ADHOC BASIS.IF MEMORY LIMITED, THE DIMENSIONS
C     CAN BE CHANGED,BUT THE SAME MUST BE CHANGED IN ALL SUBS.
C     HAVING BLANK COMMON, NAMELY DTENT,REDUCE,MERGE,PARSE
C             DCDTAB         BUFFER         BRNCH
C             CONDSN         ENTRY          EXENT
C             EXCUTD         HEADER         LIST
C             PREVNO         N              STMNT
C
C
C          IF ANY CHANGES ARE MADE IN DIMENSIONS OF THE ABOVE
C     MENTIONED VARIABLES,SEE THEIR EFFECT AT STATEMENT NOS
C     101,10000,99991  WHILE INITIALISING IN FORTAB (MAIN)
C
C                      LIST AND FUNCTION OF VARIOUS POINTERS
C
```

```
C
C      NAME      VALUE    WHERE USED       INDICATION
C---------------- - -------- ------------------  --------------
C
C
C      IFOUND    0,100    FORTAB           AFTER HAVING GOT A SUBTABLE
C                                          AT 62996,90000
C      FOUND     0                         NORMAL CARD
C      FOUND     100                       CONTINUATION CARD OF AN IF
C      FOUND     0        AFTER SEARCH     SEARCH FAILS
C      FOUND     100      AFTER SEARCH     SEARCH SUCESSFUL
C      FOUND     0       SQUEZE        IT IS A DATA OR FORMAT
C                                          DECLERATION
C      FOUND     100      SQUEZE           NOT A DATA OR FORMAT
C      INDEX     1,2,3    AFTER CALL TO PUTBRN
C      FOUND     0       PUTBRN        IF AN *AND* OR *OR* WITH
C                                          *NOT* FOUND IN LOGICAL
C                                          EXPRESSION OF AN IF
C      FOUND     100     PUTBRN        ARITH. RELATIONAL OPR.
C      FOUND     100     629 ETC       SUBTABLE FORMED,NEXT CARD
C                                          HAS A BLANK STMNT.
C      IQUIT     100                       MATCHING EXTREME ) IS FOUND
C      IQUIT     0                         EXPECTED AN .OR. OR AN .AND
C      LOGIFO    TRUE     LOGICAL IF       SAME AS IQUIT=100
       ITRM2S    0                     KEEP TRACK OF SUBSCRIPTED
C                                          VARIABLES
C      ITRM2S    0                         TERM2 IS NOT SUBSCRIPTED
C      ITRM2S    100                       TERM2 IS SUBSCRIPTED
C      IPATH     1,2      USED TO CONNECT LOG.IFS AND GOTOS
C      IPATH     0        CLASFY           CHARACTER COL10 IS A(
C      IPATH     100      CLASFY           CHARACTER COL10 IS NOT A  (
C      JPATH     1,2      CLASFY           DIFFERENT PARTS OF TREE
C      LSTCRD    0,100   CLASFY        FOR LAST CARD AS AN IFIT
C                                          IS 100
C      REFTAB    0        OUTPUT           REFERENCE TABLE NOT NEEDED.
C      REFTAB    100      OUTPUT           REFERENCE TABLE IS NEEDED
C      ISSTOP    0        LOGICAL IF       NORMAL CARD  SEE NEXT
C      ISSTOP    100     LOGICAL IF    IF A CALL EXIT OR IF(XXX)
C                                          STOP IS FOUND
C      ISTOP1    -        -            IF .GT. 1 MEANS GENERATED
C                                      STOP HAS ALREADY APPEARED ONCE
C      KTEMP     0        FORTAB           IN AND ORS  NORMAL VALUE
C      KTEMP     100     FORTAB        WHEN RELATIONOP IS MET
C                                          BEFORE
       SUBTAB                          INDEX FOR SUBTABLES.
C                                          HEADER
C      SUBDT     CLASFY   FOR NESTED IFS   T/F
C      SUBDT     PHASE2   TO PROCESS SUB DTS CORROSP TO GOTOS
C      NOTOP     0,100   LOGICAL IF    100 ONLY IF *NOT* OP
C                                          ENCOUNTERED.STORED IN HEADER
C      7.        15       23       30      38     45              60
C
C
```

```
C                                   PHASE 2
C                                   ----------
C
C
C         IPATH     0               PHASE2
C         IPATH     1              PHASE2              NEW CONDITION
C                                                 CONDITION HAS ALREADY
C         RESET     J               PHASE2             APPEARED
C                                                      KEEP TRACK OF ROWS OF TABLE
C                                                      ALREADY SCANNED ONCE
C                                   TAPE ALLOCATION
C                                   ---------------
C
C
C         TAPE UNIT 0               LISTING OF SOURCE DECK
C         TAPE UNIT 1               TABLE1
C         TAPE UNIT 2               TABLE2
C         TAPE UNIT 3               USED IN PARSE
C         TAPE UNIT 4              PARTIAL DECISION TABLE CORROSPONDING
C                                  TO LOGICAL IFS.
C
C
C         IN CASE WE ARE TIED UP ,0 CAN BE RELEASED.
C         SEE STATEMENTS 110+6,750
C**********************************************************************
          DO 10103 I=1,100
10103     RENTER(I)=0
          DO 103 RULE=1,10
          DO 102 LLL=1,10
102       EXENT(LLL,RULE)=BLANKS
103       LIST(1,RULE)=BLANKS
C         INITIALISE FOR SUBTABLES
          DO  2  JK=1,26
2         ALNUM(JK)=ALPBET(JK)
          DO  3  JK=1,10
3         ALNUM(JK+26)=NUMBER(JK)
C
          L=1
          MSTOUT=BLANKS
          IFOUND=0
C         SEE 90000 AND 62996+2
          JPREV=1
10301     FORMAT(//1H ,36A3)
          M1=1
          ROWTA1=0
          LOGIFO=.FALSE.
          SUBET=.FALSE.
          ROWTA2=0
          LSTCRD=0
          SUBTAB=0
          ISSTOP=0
          REWIND  0
          REWIND  1
```

```
      REWIND  2
      REWIND 4
      DO 31234 K=1,600
31234 BUFFER(K)=BLANK
      CARYON=100.
      ICON=1
      ISTOP1=
      PRINT 29998
29998 FORMAT(1H1,*ISN*,10X,*SOURCE  STATEMENT*///)
      IGOAL=80008
      CALL BINBCD(IGOAL,FOLLOW)
      CALL REBLBL(FOLLOW,NUMBER(10))
      I80008=FOLLOW
9999  READ 10,COLS
C     SAMPLE DECK IS READ ALSO AT 62951 AND 751
C     READ SAMPLE DECK
9998  DO 19999 K=1,JPREV
19999 BUFFER(K)=BLANK
99990 JPREV=0
      FOUND=0
      KTERM=0
      NOCRAT=0
      CALL   BINBCD(N1,NN)
      CALL   REBLBL(NN,NUMBER(10))
      BCDTAB(N1)=NN
99001 FORMAT(1H ,*NN=...*,A6///)
5     FORMAT(A5)
6     FORMAT(A6)
      DO   9 I=1,7
9     ENTRYS(I)=BLANKS
10    FORMAT(80A1)
11    FORMAT(1H ,A3,4X,80A1)
      WRITE(99,10) (COLS(I),I=1,6)
110   FORMAT(8X,30A1)
      READ(99,6) DECKEN
      IF(DECKEN.EQ.DEKEND)  GO TO 29999
C
C     ................................................................
C
C     EACH PROGRAM DECK MUST HAVE A DEKEND AT THE END ,PUNCHED
C     FROM COL.1. THIS IS A CONTROL CARD FOR ..LOGITRAN..
C     THE REMAINING COLS. OF THIS CARD SPECIFY THE OPTIONS.
C                  COL  10   TABLEA
C                  COL  20   TABLEB
C                  COL  30   PARSE
C
C          FOR PARSING ONLY, THE FIRST CARD SHOULD BE  DEKEND
C     WITH  PARSE IN COLS. 30 TO 34, FOLLOWED BY DECISION
C     TABLE TO BE PARSED IN  5A3,16A4 FORMAT
C
C     ................................................................
      IF(COLS(6).NE.BLANK.AND.COLS(6).NE.NUMBER(10).OR.COLS(1).E
```

```
      10.C)  NN=BLANKS
       PRINT 11,NN,COLS
       IF(COLS(1).EQ.C)  GO TO 9999
       WRITE(?,12) N1,COLS
12     FORMAT(I5,3X,80A1)
       IF(COLS(6).NE.BLANK.AND.COLS(6).NE.NUMBER(10).AND.LSTCRD.E
      1Q.1??)  GOTO 9999
C      THIS IS THE CASE  OF AN IF WHERE THE     ORDER,FOLLOW HAVE
C      APPEARED ON THE MAIN CRD BUT ARGUMENT LIST OR I/O IS
C      APPEARING ON THE NEXT CONTINUATION CARD
       IF(COLS(6).NE.BLANK.AND.COLS(6).NE.NUMBER(10))GO TO 9999
C--------------- - ------------------------- -------------
C
C      CONTINUATION CARDS WHICH ARE NOT FOR AN $$$ IF $$$  ARE OF
C      NO INTEREST  TO US.  .. IF.. CONTINUATION CARDS WILL BE
C      READ BY A SEPARATE READ  STATEMENT AT THE APPROPRIATE
C      PLACE. LOOK STMNT. 750
C
C--------------- ---------- ------------------ -------------
       CALL  SQUEZE(COLS,JPREV)
       IF(FOUND.EQ.0)  GO TO 9997
90000  CONTINUE
       WRITE(99,10)(COLS(I),I=1,5)
       READ(99,5) STMNNO
       IF(STMNNO.NE.BLANKS)  GO TO 9995
       IF(SUBDT.AND.FOUND.EQ.100) STMNNO=ITEMP
C
C      FOR BLANK COLS. 1-5 ,WHY NOT KEEP AS CONCATNATED WITH S.NO
C                           FEB. 1969
C
       GO TO 9993
9997   N1=N1+1
C      SHALL SEE IF WE ARE TO LIST IT OR NOT
       GO TO 9999
9995   CALL  REMLBL(STMNNO,BLANK)
       MSTMNT=STMNNO
9993   UPLIMT=6
       SUBDT=.FALSE.
C      SEE COMMENT AFTER 62955 AND BEFORE 62996
       INITAL=10
C *************************************************************
C *
C *
C                              CLASFY
C *
C *
C *************************************************************
C      THIS IS THE BEGINING OF CLASIFY ,WHICH DOES CLASSIFICATION
C      OF STATEMENTS
       WRITE(99,300)  BUFFER(7),BUFFER(8)
146    FORMAT(1H ,14A6)
300    FORMAT(6A1)
```

```
      READ(99,302) BUF78
302   FORMAT(A2)
C     THERE IS NO NEED TO KEEP   $$DO$$ IN THE IF  BELOW
C     9997 IS THE CASE OF $$DATA$$ AND $$FORMAT$$ DECLERATIONS
C     SEE STATEMENT 62996+1,......+3
      IF(SUBDT.AND.IFOUND.EQ.100)  STMNNO=ITEMP
      IFOUND=0
      IF(BUF78.EQ.GO.OR.BUF78.EQ.DO.OR.BUF78.EQ.IF.OR.BUF78.EQ.S
     1T)  GOTO 315
      IF(BUF78.EQ.RE.OR.BUF78.EQ.CA)  GOTO 315
310   IF(STMNNO.EQ.BLANKS)  GO TO 306
C     THIS WAY MOST OF THE DECLERATIONS WILL NOT GET LISTED IN
C     IN TABLE 2.
      TABLE=2
      ROWTA2=ROWTA2+1
      GO TO 316
306   GOTO 9997
315   TABLE=1
      IF(BUF78.EQ.DO)  GO TO 400
C     THERE IS NO NEED TO ROUTE IT TO 400 AS DO NEED NOT BE PROC
C    /ESSED
      WRITE(99,300) (BUFFER(K),K=7,9)
      READ(99,303) BUF79
303   FORMAT(A3)
      IF(BUF79.EQ.IFLEFT)  GO TO 500
      WRITE(99,300) (BUFFER(K),K=7,10)
      READ(99,304)  BUF710
304   FORMAT(A4)
305   FORMAT(A5)
      IF(BUF710.EQ.GOTO4)  GO TO 800
      IF(BUF710.EQ.STOP4)  GO TO 900
      IF(BUF710.EQ.CALL)  GOTO 960
      WRITE(99,300)(BUFFER(K),K=7,12)
      READ(99,6) BUF712
      IF(BUF712.EQ.RETURN)  GOTO 940
      GO TO 310
220   FORMAT(1H ,7A6,A6,2X,8(5X,A5))
316   BRNCH1=DASH
      BRNCH2=DASH
      BRNCH3=DASH
      LSTCRD=0
      WRITE(2,319) (BUFFER(I),I=7,42),NN,STMNNO,BRNCH1,BRNCH2,BR
     /NCH3
319   FORMAT(36A1,A6,2X,8(5X,A5))
320   FORMAT(7A6,A6,2X,8(5X,A5))
321   FORMAT(1H ,80A1)
      N1=N1+1
      GO TO 9999
C ***********************************************************************
C *
C*                                      PROCESSING FOR  DO
C *
```

```
C***********************************************************************
C      DO NEED NOT BE PROCESSED. CHANGING   400   INITAL=9 TO 400
C      GOTO 316
400    GOTO 316
C      SEE NOTE BEFORE 400  GOTO 316
430    PRINT  435,NN,(BUFFER(I),I=1,80)
       LOGIFO=.FALSE.
       SUBOT=.FALSE.
       IPATH=200
C      RESET ALL POINTERS
435    FORMAT(1H ,A5,*THE FOLLOWING STATEMENT SEEMS TO BE WRONG I
     /N SYNTAX
     . OR IT HAS NOT BEEN TAKEN CARE OF IN THIS IMPLEMENTATION*/
     /1H ,80A1
     2)
       N1=N1+1
       CAPYON=0.
       LSTCRD=0
       GO TO 9999
C *********************************************************************
C *
C *                              PROCESSING FOR AN IF
C *
C *********************************************************************
C                   C L A S S I F I C A T I O N   OF  I F S
C
C-------------- ---------- ------------------+ -- ------- ---
C
C       IF(XXX.OP.XXX)  GO TO NNN             TYPE1
C       IF(XXX-XXX) NN,NN,NN                  TYPE2
C       IF(X)(X)   NN,NN,NN                   TYPE3
C       IF(XXX.OP.XXX)  EXPRESSION            TYPE4
C       IF(XXX.OP.XXX)   CALL SUB             TYPE5
C       IF(XXX.OP.XXX.AND.XXX.OP.XXX......)   LOGICAL IF
C       IF(XXX.AND..NOT.XXX)                  LOGICAL IF WITH NOT
C       IF(.NOT.XXX)                          LOGICAL IF WITH NOT
C       IF(XXX)XXX                            IF OF DECLARED LOGICAL
C       IF(XXX.GT.XXX.AND.XXX)XXX             IF OF DECLARED LOGICAL
C----------------------------- ------------------- -------------
500     INITAL=10
        LSTCRD=100
        JPATH=0
        IPATH=200
        LOGIFO=.FALSE.
        GOTOP=0
        BRNCH1=BLANKS
        BRNCH2=BLANKS
        BRNCH3=BLANKS
        RULE=1
        NOOFOP=0
        NOCONT=0
```

```
      IROUTE=200
      TERM1=BLANKS
      TERM2=BLANKS
      OPRATR=BLANKS
      LLL=1
      DO  5000 K=1,7
5000  ENTRYS(K)=BLANKS
C     LLL WILL DECIDE THE CONDITIONS IN A LOGICAL IF CONTAINING
C     AND/OR
      UPLIMT=63
C     IF STATEMENT STAPTING WITH COL.7(BLANKS SQUEZED OUT) CAN
C     HAVE A MAX. OF 63 CHRS. UPTO ) ,IF IT DOES NOT HAVE CONT-
C     NUATION CARD CONTAINING A PART OF CONDITION
      ISTART=10
      CALL  GETENT( LONGEX,MAXNOP)
      IF(FOUND.EQ.0)  GO TO 750
      JLEFT=INITAL-1
C-------------------------------------------------------------------------
C
C     THOUGH A WELL FORMED EXPRESSION, THAT IS MATCHING
C     PARANTHESIS FOUND,YET IT IS POSSIBLE THAT A CONTINUATION
C     CARD MAY BE THERE.WE ARE PERTICULARLY INTERESTED IN GOTO
C     OR CALL OF AN IF MENTIONED NEXT CARD
C---------------------------------+---------------------------------------
      IF((FINAL+1+4).GT.JPREV)  GO TO 50520
C
501   IJKPRE=FINAL
      INITAL=FINAL+2
C     ADDITIONAL FOR )
50100 INPLS3=INITAL+3
      WRITE(99,300) (BUFFER(K),K=INITAL,INPLS3)
      READ(99,304) ORDER
      IF(ORDER.EQ.GOTO4) TYPE=1
      IF(ORDER.EQ.CALL) TYPE=5
      IF(ORDER.EQ.CALL.OR.ORDER.EQ.GOTO4)  GO TO 50010
      IF(ORDER.EQ.STOP4)  GOTO 50518
      IF(ORDER.EQ.RETU)  GOTO 50517
50516 UPLIMT=6
      CALL SEARCH(COMMA,NUMBER,10,BRNCH1)
      IF(FOUND.EQ.0)  GOTO 50519
      TYPE=23
      GO TO 502
50517 INPLS3=INPLS3+2
      WRITE(99,300) (BUFFER(K),K=INITAL,INPLS3)
      READ(99,6) ORDER
      IF(ORDER.EQ.RETURN)  GOTO 50518
      GOTO 50516
```

```
50518 TYPE=4
      FOLLOW=I88888
      ENTRYS(7)=ORDER
      GOTO 502
50519 TYPE=4
      ENTRYS(7)=IHARTH
      FOLLOW=STARS
      GOTO 502
50520 FOUND=100
      READ 10,COLS
C     READS A PROBABLE CONTINUATION CARD
      IF(COLS(6).EQ.BLANK.OR.COLS(6).EQ.NUMBER(10))  GO TO 50525
      WRITE(6,12) N1,COLS
50522 FORMAT(1H ,*IT IS A CONTINUATION CARD OF CARDNO=*,A6)
      CALL SQUEZE(COLS,JPREV)
      GOTO501
50525 WRITE(99,10) COLS
C     IT IS NOT A CONTINUATION CARD
      N1=N1+1
      GOTO99990
50524 ISSTOP=100
      FOLLOW=I88888
      GOTO 5020
50526 ENTRYS(6)=CALL
      ENTRYS(7)=FOLLOW
      GOTO 5020
50010 INITAL=IPPLS3+1
      UPLIMT=6
      CALL  SEARCH(BLANK,ALNUM,36,FOLLOW)
C     IN THE CASE OF  IF(...) CALL SUB(..)  SEARCH WILL FAIL.SO
C    /DO THE
C     NEEDFUL
      IF(FOUND.EQ.0.AND.ORDER.EQ.CALL)  CALL SEARCH(LEFTP,ALNUM,
     136,FOLLOW)
502   FINAL=IJUPRE
      IF(ORDER.EQ.STOP4.OR.ORDER .EQ.RETURN.OR.ORDER.EQ.CALL.AND
     1FOLLOW.EQ.EXITT)  GOTO 50524
50527 IF(ORDER.EQ.CALL)  GOTO 50526
5020  INITAL=10
C
C     IF(.NOT.XXXXXXXXXXX)XXXX
      IF(BUFFER(10).EQ.PERIOD.AND.BUFFER(11).EQ.ALPBET(14).AND.B
     /UFFER(12
     1 ).EQ.ALPBET(15).AND.BUFFER(13).EQ.ALPBET(20).AND.BUFFER(1
     14).EQ. PERIOD)  GOTO 7891
C
C
```

```
5021   IF(MAXNUP.LT.2)   GOTO 503
       GO TO 030
7891   NOTOP=100
C      NOTOP IS POINTER FOR ENOTS OPERATOR
       INITAL=INITAL+5
       GOTO 5030
C
C
503    CONTINUE
       NOTOP=0
C      INITIALISE NOTOP FOR NEXT ITERATION
780    J=INITAL
       IF(BUFFER(J).EQ.PERIOD.AND.BUFFER(J+1).EQ.ALPBET(14).AND.B
      /UFFER(J+
      12).EQ.ALPBET(15).AND.BUFFER(J+3).EQ.ALPBET(20).AND.BUFFER(
      1J+4).EQ.PERIOD)   GOTO 7891
5030   CALL SEARCH(PERIOD,ALNUM,36,TERM1)
       IF(FOUND.EQ.0)   GO TO 550
       ITEMP1=TERM1
C      SAVE TERM BECAUSE IF NEXT SEARCH FAILS THEN CONTENTS OF
C      TERM SHOULD NOT GET CLOBBERED
       IFINAL=FINAL
       CALL   SEARCH(MINUS,ALNUM,36,TERM1)
       IF(FOUND.EQ.100)   GO TO 552
C      THIS IS FOR A CASE LIKE      IF(35.23-A) 10,20,23
C      SEE COMMENT AFTER 575
       TERM1=ITEMP1
C*****************************************************************
C                                   TYPE1 IF
C*****************************************************************
       TYPE=1
       UPLIMT=5
C      THIS WILL COVER AND ALSO.DOES THE UPLIMT REALLY MATTER,
C      SCANNING IS FROM LEFT TO RIGHT,SO SHOULD NOT.
       INITAL=IFINAL+1
       CALL   SEARCH(PERIOD,ALPBET,26,OPRATR)
       IF(FOUND.EQ.0)   GOTO 570
       INPLS3=INITAL+3
       IF(BUFFER(INITAL+1).EQ.ALPBET(1))   INPLS3=INPLS3+1
C      THIS WILL TAKE CARE OF AND OPERATOR
       WRITE(99,300)   (BUFFER(I),I=INITAL,INPLS3)
       READ(99,304)   OPRATR
       IF(BUFFER(INITAL+1).EQ.ALPBET(1))   READ(99,305) OPRATR
505    INITAL=FINAL+1+1
C      LAST VALUE OF FINAL IS UPTO AND EXCLUDING . OF .OP.
       ISTMN=505
C      DELETE THE ABOVE COMMENT CARD IN FINAL FORM
       UPLIMT=6
       J=INITAL
       ISTMN=505
       CALL PUTBRN(OPRATR,FOLLOW,BRNCH1,BRNCH2,BRNCH3,NN)
```

```
      IF(FOUND.EQ.1.OR.FOUND.EQ.2)  GOTO(781,782), FOUND
      IF(FOUND.EQ.3)  GOTO 480
      CALL SEQUNC(LONGEX,M,JPREV)
      IF(LONGEX(1).EQ.RIGHTP.OR.LONGEX(1).EQ.PERIOD.AND.LONGEX(2
     1).EQ.RIGHTP)  GOTO 50501
      IF(LONGEX(1).EQ.PERIOD.AND.LONGEX(2).EQ.PERIOD)  GOTO 5050
     12
      GOTO 480
C     480  ILL INCLUDE CASES WHERE + SIGN IS USED INSIDE AN IF
50501 LOGIFC=.TRUE.
      IROUTE=1
      CALL SEARCH(RIGHTP,ALNUM,36,TERM2)
      GOTO 50510
50502 LOGIFC=.FALSE.
      IROUTE=2
      CALL SEARCH(PERIOD,ALNUM,36,TERM2)
      IF(BUFFER(FINAL+2).NE.ALPBET(1).AND.BUFFER(FINAL+2).NE.ALP
     1BET(15))  GOTO 50503
      GOTO 50510
C
C
50504 FINAL=FINAL+1
C     THIS IS THE CASE WHEN DECIMAL PT. DOES NOT HAVE NUMERALS F
C    /OLLOWING
C     IT
      GOTO 50505
50503 ISAVE=INITAL
      IF(BUFFER(FINAL+2).EQ.PERIOD)  GOTO 50504


      CALL SEARCH(PERIOD,NUMBER,10,TERM2)
50505 WRITE(99,300)  (BUFFER(IJ),IJ=ISAVE,FINAL)
      READ(99,6)  TERM2
      GOTO 50510
50510 INITAL=FINAL+2
      IQUIT=0
      ITRM2S=0
C     DELETE THESE I.E. ITRM2S AND IQUIT IN FINAL FORM
      IF(BOBDT)  GOTO 780
C     ALREADY BRANCHES HAVE BEEN FOUND
5060  IF(ORDER.EQ.GOTO4)  GOTO 509
      IF(ORDER.EQ.CALL)  GO TO 508
      IF(ORDER.EQ.STOP4.OR.ORDER.EQ.RETURN)  GOTO 780
C*****************************************************************
C                                        TYPE4  IF
C*****************************************************************
50701 BRNCH1=STARS
      BRNCH2=STARS
```

```
      TYPE=4
      ENTRYS(6)=CPRATR
C     THIS WILL HELP FOR IFS HAVING A SINGLE CONDITION ONLY. FOR
C    / SUBDTS
C     (LOCDT)   IT WILL NOT SERVE ANY PURPOSE
      BRNCH3=STARS
      GO TO 780
50705 IPATH=123
      J=INITAL
      ASSIGN 503 TO LOGIF
      GOTO(781,782,788), FOUND
C     THIS WILL TAKE CARE OF CASEC LIKE  IF(A.OR./AND/NOT.B)XXX
5070  ITEMP=188833
C     5070 MAY HAVE TO BE DELETED IN THE LONG RUN
      TYPE=45
      GOTO 780
508   TYPE=5
      IF(FOLLOW.EQ.188888)           GOTO 780
C     BY ... BECAUSE OF CALL EXIT FOLLOW =88888
      GOTO 50701
C     CHECK UP THE LOGIC
C***************************************************************************
C                                         REST OF TYPE1 IF
C     **********************************************************************
509   CONTINUE
      UPLIMT=24
C     IN A TYPE4 IF,WE CAN HAVE GOTO(XX,XX,XX)=XX     . = IS AT 2
C    /4
C
C     COMPUTED AND ASSIGNED GOTO ARECALSO POSSIBLE
C
      CALL  SEARCH(EQUALS,ALNUM,36,GOAL)
C     THE ABOVE SEARCH SEEMS ILLOGICAL
C
C     IF(FOUND.EQ.100) GOTO 50701
C     IF(BUFFER(INITAL+1).EQ.LEFTP)  GOTO ....
C     CALL SEARCH(COMMA,ALNUM,36,INDEX)
C     IF(FOUND.EQ.100) GOTO......
      IF(FOUND.EQ.0)   GOTO 780
      GOTO 50701
C     ACTION WILL BE TO GOTO TABLE CORROSP. TO COMP.GOTO,SET POI
C    /NTER,
C     IT BEFORE ENTERING LIST
560   ENTRYS(1)=STOP4
      ENTRYS(2)=SLASH
      ENTRYS(3)=RETURN
C     WE ENTER 560 FROM 5599 AFTER ENTERING PHASE2 AND BACK
C     TO PHASE1
      DO 5601 KLM=4,7
5601  ENTRYS(KLM)=BLANKS
      NNN=900
```

```
      CALL BINBCD(NMN,NN)
      CALL REMLBL(NM,NUMBER(10))
C--------------  - -------- ----------------   - ------  ---
C
C     STOP WILL BE A GENERATED ENTRY.WE ARE SURE THAT STMNNO
C     CAN NOT  APPEAR  IN A STATEMENT BECAUSE IT IS NOT ALLOWED
C     BY FORTRAN-IV.
C
C     .... NO , YOU MAY HAVE TO CHANGE IT..
C     SIMILARLY NM=400 IS AN ASSIGNED VALUE                 -
C
C--------------    -------- ----------------  -- ----------
      STMNNO=ITEMP
      BRNCH1=DASH
      BRNCH2=DASH
      BRNCH3=DASH
      ISSTOP=2
      GO TO 652
5566  N1=N1-1
C     N1 WAS INCREMENTED BUT HERE IT IS NOT NEEDED
      GO TO 9999
5577  FOUND=2
      GO TO 5599
5588  FOUND=1
5599  ITEMP=188888
      GOTO 560
C****************************************************************
C                                   TYPE2 IF  TYPE3 IF
C****************************************************************
550   CALL  SEARCH(MINUS,ALNUM,36,TERM1 )
      IF(FOUND.EQ.0)  GO TO 570
C     IF 570 MUST BE TYPE3 IF
552   TYPE=2
      INITAL=FINAL+2
C     ONE IS NORMAL AND THE OTHER IS FOR -
      OPRATR=DASH
      CALL  SEARCH(RIGHTP,ALNUM,36,TERM2)
      IF(FOUND.EQ.0)  GO TO 480
C     CHECK UP SYNTAX
      LOGIFO=.TRUE.
555   INITAL=FINAL+2
      IF(BUFFER(INITAL).EQ.EQUALS)  GO TO 310
C     IT MUST     BE SIMILAR TO IF BUT NOT AN IF
      CALL  SEARCH(COMMA,NUMBER,10,BRNCH1)
      INITAL=FINAL+2
      CALL  SEARCH(COMMA,NUMBER,10,BRNCH2)
      INITAL=FINAL+2
      CALL  SEARCH(BLANK,NUMBER,10,BRNCH3)
      IF(JPATH.EQ.1)  GOTO 652
C     THIS WILL HELP FOR ARITHMATIC EXPR. OF GT THAN 6 CHARS.
      LOGIFO=.TRUE.
```

```
      GO TO 780
556   ENTRYS(1)=TERM1
      ENTRYS(2)=DASH
      ENTRYS(3)=TERM2
      ENTRY(LLL,1)=TERM1
      ENTRY(LLL,3)=TERM2
      ENTRY(LLL,2)=DASH
      ISTMM=556
      JSTMNT=556
      EXENT(LLL,RULE)=OPRATR
      GOTO 558
557   ENTRYS(1)=TERM1
      ISTMM=557
      JSTMNT=557
      ENTRYS(2)=BLANKS
      ENTRY(LLL,2)=BLANKS
      ENTRY(LLL,3)=BLANKS
      ENTRYS(3)=BLANKS
      ENTRY(LLL,1)=TERM1
      EXENT(LLL,RULE)=T
C     THIS IS THE CASE OF OF AN IF HAVING DECLARED LOGICAL VARIA
C    /BLES
558   IPATH=5
5582  ENTRYS(4)=BLANKS
      ENTRYS(5)=BLANKS
      ENTRY(LLL,4)=BLANKS
      ENTRY(LLL,5)=BLANKS
      LOGIFO=.TRUE.
      ASSIGN 5583 TO LOGIF
      GOTO 62906
C*******************************************************************
C                                      TYPE3 IF
C                       AND LOGICAL TYPE 3 IF
C*******************************************************************
570   CALL  SEARCH(RIGHTP,ALNUM,36,TERM1)
      IF(FOUND.EQ.0)  GO TO 576
      IF(NOTOP.EQ.100)  GOTO 5762
      IF(OPRATR.EQ.ANDD.OR.OPRATR.EQ.ORR)  GOTO 557
      ENTRYS(1)=TERM1
      IF(LLL.NE.1)  GOTO 557
      IF(TYPE.EQ.1)  GOTO 792
      IF(TYPE.EQ.4.OR.TYPE.EQ.5)  GOTO 792
C     THIS WILL TAKE CARE OF LOGICAL NOT OPRATR
      TYPE=3
      GO TO 555
C     TO GET THE BRANCHES
C-------------------------------------------------------------------
C     THIS IS THE CASE OF ARITHMATIC IF. COMPARISON IS DONE
```

```
C       WITH EPSILON.TEST FOR BOTH
C       THIS WILL TAKE CARE OF CASES WHERE TERM IS LONGER THAN 6
C       CHARACTERS
C---------------  - --------- ------------------  -- ------- ---
C       THIS WILL TAKE CARE OF CASES WHERE TERM IS LONGER THAN 6 C
C      /H.
576     KMAX=(IJKFIN-INITAL+1)/6+1
        IF(((IJKFIN-INITAL+1)-((IJKFIN-INITAL+1)/6)*6).EQ.0) KMAX=
       /KMAX-1
        JPATH=1
        FINAL        =IJKPRF
5760    DO 5761 I=1,KMAX
5761    ENTRYS(I)=LONGEX(I)
        GO TO (555,5822),JPATH
C
5762    ENTRYS(1)       =NOT
        ENTRYS(2)=TERM1
        ENTRYS(3)=BLANKS
        ENTRY(LLL,3)=BLANKS
        EXENT(LLL,RULE)=T
        ENTRY(LLL,1)=NOT
        ENTRY(LLL,2)=TERM1
        ISTMNT=5762
        JSTMNT=5762
        IPATH=4
        GOTO 5532
C******************************************************************
C
C       NESTED LOGICAL IFS WITHOUT EXTRA ( OR )
C
C******************************************************************
600     IF(.NOT.SUBDT)   INITAL=10
6000    IF(MAXNOP.GT.1)   GO TO 680
        GOTO 593
C       FEB. 19,1969
C       THIS IS TO TAKE CARE OF ANDS ETC IN LOGICAL OPS
C---------------    --------- ------------------  -- ------------
C
C       THIS PATH FOR STATEMENTS CONTAINING NO ( OR ), NITHER
C       FOR SUBSCRIPTED VARIABLES NOR FOR THE SAKE OF CLARITY
C       THE CHARACTER SET WITHIN ( ) IS MORE THAN 42 CHRS.
C
C---------------    --------- ------------------  -- ------- ---
625     CALL  SEARCH(PERIOD,ALNUM,36,TERM1    )
        JSTMNT=625
        IF(FOUND.EQ.0)  GO TO 480
        IF((FINAL-INITAL+1).GT.6)  GOTO64010
```

```
C     THIS WILL DEAL WITH CASES LIKE LC+ICC+2 IN TERM
C     DATA CARD FOLLOWS
C     IF(KOCT(ICD).NE.0.AND.KOCT(ICD+3).EQ.10.AND.LC+ICC+2.EQ.IE
C    /ND)
620   INITAL=FINAL+1
      INPLS3=INITAL+3
      IF(BUFFER(INITAL+1).EQ.ALPBET(1))  INPLS3=INPLS3+1
      WRITE(99,300)   (BUFFER(I),I=INITAL,INPLS3)
      READ(99,304)   OPRATR
      IF(BUFFER(INITAL+1).EQ.ALPBET(1))  READ(99,305) OPRATR
      INITAL=INPLS3+1
      J=INITAL
C     THIS WILL BE LATER USED BY 781,782,788 ETC IF NEEDED
      CALL   SEARCH(RIGHTP,ALNUM,36,TERM2)
      IF(FOUND.EQ.100.AND.(FINAL-INITAL+1).LE.6)   GOTO650
C
620   CALL  SEARCH(PERIOD,ALNUM,36,TERM2)
      IF(FOUND.EQ.0) GOTO6300
C     SUCESS IN FINDING INDICATES NESTED LOGICAL IF
C     6300 MUST BE DUE TO CASE IN WHICH TERM CONTAINS A +,- SIGN
C     AND NO ( OR )
C
630   ASSIGN 503 TO LOGIF
C     FEB 18,1969
C
      JSTART=630
      ISTRM=630
      GOTO 62900
C     IT MUST BE DUE TO LOGICAL EXPRESSIONS CONTAINING $SAND,OR,
C    /NOT$$
C
C---------------     --------  ------------------   --  -----------
C
C                         NESTED LOGICAL IFS
C                         FEB.,1969
C
C---------------  -  --------  ------------------   --  -------  ---
C     SUCESS IN FINDING INDICATES NESTED LOGICAL IF
629   LLL=1
      RULE=1
62900 ENTRY(LLL,1)=TERM1
      ENTRY(LLL,2)=DASH
      ENTRY(LLL,3)=TERM2
      ENTRY(LLL,4)=BLANKS
      ENTRY(LLL,5)=BLANKS
62905 EXENT(LLL,RULE)=OPRATR
      JSTANT=62900
62906 IF(LOGIFO.AND.LLL.EQ.1)   GOTO 62916
C     SEE NOTE AT 62950
      SUBDT=.TRUE.
```

```
      IF(LOGIFO)  GOTO 62950
      IF(IPATH.EQ.4)  GOTO 62917
C     IPATH=4 IS FOR IFS CONTAINING DECLARED LOGICAL VARIABLES.
C     SEE 7815)
62910 INITAL=FINAL+1
      IF(BUFFER(INITAL+1).EQ.ALPBET(1))  GOTO 62920
C     FAILURE IS FOR $$OR$$
      INPLS3=INITAL+3
      WRITE(99,300) (BUFFER(K),K=INITAL,INPLS3)
      READ(99,304)     LOGOPR
      RULE=RULE+1
62915 INITAL=INPLS3+1
62914 ISTMN=62915
      LLL=LLL+1
78954 GOTO LOGIF (503,403)
62917 IF(OPRATR.EQ.OPR) RULE=RULE+1
      GOTO 62914
C
62916 LOGIFO=.FALSE.
      IF(IPATH.NE.4) IPATH=3
C     IPATH =4 FOR A $NOT$ OP.
      GOTO 62993
C     BEFORE GOING TO 652 ,MUST RESET LOGIFO,AND CLEAR LOCATIONS
62920 INPLS3=INITAL+4
C     THIS PATH FOR $$AND$$ OPRATOR
      WRITE(99,300) (BUFFER(K),K=INITAL,INPLS3)
      GOTO 62915
62950 LOSTFO=LLL
C     LOSTFO IS NO. OF SUBTABLE ROWS
      SUBTAB=SUBTAB+1
C     THERE IS A SLIGHT DIFFERENCE THE WAY THINGS ARE DEALT FROM
C     68350 AND 625.  FROM 625 WE GET TO 629 ONLY IF .AND.,.OR.
C     IS FOUND BUT FOR 68350 THAT IS NOT THE CASE.
C     SO HERE IF LOGIFO IS .TRUE. AND
C     LLL=1, WE MUST PUT IN MAIN TABLE I.E. ROUTE IT 652,653 ETC
C     BEFORE GOING TO 652 ,MUST RESET LOGIFO
C     SEE CONDITION TESTED BET. 62905 AND 62910
      IRULES=RULE+1
C     +1 TO TAKE CARE OF ELSE RULE
62951 READ 10,COLS
      IF(COLS(1).EQ.C)  GOTO  62951
C     INCREMENT THE CARD SERIAL NO.
C     NO NEED IF AT 62992 ONE KEEPS FOR GOTO 653,653,652
C     READ SAMPLE DECK NEXT CARD TO FIND COLS 1-5
      WRITE(99,10) (COLS(I),I=1,5)
      READ(99,5) ITEMP
C     SEE IF STMNT HAS A NON BLANK CHARACTER
      IF(ITEMP.EQ.BLANKS)  GOTO 62995
C     COLS 1-5 ARE NON BLANK,BUT THERE MAY BE PRECEDING BLANKS
C     SO CALL REMLBL
62955 DO 62960 I=1,RULE
```

```
62960 LIST(1,I)=FOLLOW
      LIST(1,IRULES)=ITEMP
      IPATH=1
C     IPATH=1  UTILIZED AT 6532.THIS IS TO KEEP TRACK THAT CARD
C    /ALREADY
C     READ
C     FOR IFS IPATH=1,FOR GOTOS (ASSIGNED AND COMPUTED) IPATH=2
C    /(8700)
C     FOR AN IF HAVING SINGLE CONDITIONI.E.NO SUBTABLE,IPATH=3
62965 IF(STMNNO.NE.BLANKS)  GOTO 62975
      WRITE(99,62996) AS,NN
      READ(99,5)  STMNNO
62975 HEADER(SUBTAB)=STMNNO
C     MAKE THE FORMAT MORE ELEGANT
      ENTRYS(1)=LOGDT
      ENTRYS(2)=BLANKS
      ENTRYS(3)=BLANKS
      ENTRYS(4)=BLANKS
      ENTRYS(5)=BLANKS
      BRNCH1=ITEMP
      BRNCH3=ITEMP
      BRNCH2=FOLLOW
      IF(ENTRYS(6).NE.CALL)  ENTRYS(6)=BLANKS
      WRITE(1,320) ENTRYS,NN,STMNNO,BRNCH1,BRNCH2,BRNCH3
      ROWTA1=ROWTA1+1
      WRITE(4,62985) HEADER(SUBTAB),NOSTRO,IRULES
62985 FORMAT(A5,I4,I4)
      DO 62990 I=1,LLL
62990 WRITE(4,62987) (ENTRY(I,IJK),IJK=1,5),(EXENT(I,JRULE),JRUL
     1E=1,IRULES)
62987 FORMAT(5A6,1X,16A6)
62989 FORMAT(*GO TO*,26X,16A6)
C     IS THIS 26X WASTE FUL OF TIME ,IF IT IS,CHANGE IT
      WRITE(4,62989) (LIST(1,JRULE),JRULE=1,IRULES)
C     SUBDT IS NOT INITIALISED HERE WILL BE DONE AFTER ASSIGNING
C     THE LABEL FOR THE NEXT CARD
62993 LOGIFC=.FALSE.
C     INITIALISE LOCATIONS
      DO 62991 K=1,LLL
      DO 62991 I=1,7
C     ENTRYS(I)=BLANKS            INITIALISE AFTER 653
62991 ENTRY(K,I)=BLANKS
      DO 62992 I=1,IRULES
C     EXENT(K,I)=BLANKS           INITIALISE AFTER 653
62992 LIST(1,I)=BLANKS
      GOTO(653,653,652,792),IPATH
C     FOR 792 PROPER ROUTING WILL BE DONE AFTER MODIFICATIONS(SU
C    /BST,.)
C     SEE COMMENT BEFORE 62965
```

```
      GOAL=FOLLOW
      CALL  PUTBRN(OPRATR,GOAL,BRNCH1,BRNCH2,BRNCH3,NN)
      ISTMN=650
      IF(FOUND.EQ.100) GOTO  62900
      INDEX=2
      GOTO(781,782,788,480),FOUND
C     IT MUST BE DUE TO LOGICAL EXPRESSIONS CONTAINING *AND*,*OR
C    /*,*NOT*
6500  IF(LLL.EQ.1)  GOTO652
      STMNNO=DITTO
      IF(LOGOPR.EQ.ORR)       STMNNO=NEXT
652   WRITE(1,320) ENTRYS,NN,STMNNO,BRNCH1,BRNCH2,BRNCH3
      ROWTA1=ROWTA1+1
653   N1=N1+1
      DO 6531 K=1,7
6531  ENTRYS(K)=BLANKS
      DO 6532 K=1,10
      DO 6532 I=1,10
6532  EXENT(K,I)=BLANKS
      IF(ISSTOP.EQ.200)  GOTO 31111
      IF(IPATH.EQ.1.AND.SUBDT)  GOTO 9998
C     SEE 62960+......
C     NEXT CARD HAS ALREADY BEEN READ AT 62950 FOR PATH1
      GO TO 9999
654   PRINT 6541,IQUIT,FINAL,BUFFER(FINAL)
6541  FORMAT(/1H ,*I NEVER EXPECTED THIS CONDITION*/1H ,*IQUIT=*
     1,I3,* FINAL=  *,I3,*  BUFFER(FINAL)=  *,A2/)
      GOTO 652
C*****************************************************************
C                                  NESTED LOGICAL IFS
C                                  WITH SUBSCRIPTED VARIABLES
C*****************************************************************
680   LONG=0
      ITRM2S=0
      IQUIT=0
      DO 68001 I=1,20
68001 LONGEX(I)=BLANK
      UPLIMT=JPREV
6800  IF(BUFFER(INITAL).NE.LEFTP )  GO TO 6880
      JLEFT=INITAL
6801  LPATH=0
      INITAL=INITAL+1
6805  CALL GETENT(LONGEX,MAXNOP)
      IF(LPATH.EQ.0)  FINAL=FINAL+1
      LENGTH=FINAL-INITAL+1
68051 FORMAT(1H ,*INITAL=*,I3,*FINAL=*,I3)
      KFINAL=FINAL
      ISAVE=INITAL
```

```
68052 DC 6810 I=INITAL,FINAL
      IF(BUFFER(I).EQ.PERIOD)  GO TO 6830
      IF(BUFFER(I).EQ.MINUS)  GO TO 6820
6810  CONTINUE
      GO TO 480
6820  INITAL=I+1
      IF(TYPE.NE.2.AND.TYPE.NE.3)  GOTO 68052
C     THIS WILL TAKE CARE OF CASES WHERE ..-.. SIGN IS A PART OF
C    / SUBSCRI
C     T.  EXAMPLE-  IF(SW(NEST-1,K2).NE.XXX)       XXXXX
C     THIS WILL INCLUDE CASES WHERE$-$ IS A PART OF ARITH.EXPRES
C    /SION
      UPLIMT=28
      CALL  SEARCH(LEFTP,ALNUM,36,VARIAB)
      IF(FOUND.EQ.100)  ITRM2S=100
C----------------  -------------------------  -- ----------
C     ITRM2S IS A POINTER TO KEEP TRACK OF SUBSCRIPTED VARIABLES
C     IN TERM2 IT IS 0 NORMALLY UNLESS TERM2 IS SUB. WHEN IT IS
C     100
C---------------   -------- -----------------  -- ----------
      CALL SEARCH(RIGHTP,ALNUM,36,TERM2)
      IF(KFINAL.GT.FINAL)  GOTO 68221
      IQUIT=100
      INITAL=10
      IF(LPATH.EQ.0)  INITAL=11
C     THIS WILL NOT INTERFERE WITH ANYTHING BECAUSE WITH A - SIG
C    /N NESTED
C     IFS ARE NOT POSSIBLE
      IF(ITRM2S.EQ.100) FINAL=FINAL+1
C     THIS IS TO ACCOUNT FOR AN EXTRA )
      WRITE(99,10) (BUFFER(KK),KK=INITAL,FINAL)
      FULWRD=LENGTH/6
      REMAIN=LENGTH-FULWRD*6
      CALL  BINBCD(FULWRD,FARMAT(2))
      CALL  BINBCD(REMAIN,FARMAT(4))
      IF(REMAIN.NE.0)  FULWRD=FULWRD+1
      IF(FULWRD.GT.5)  GOTO 480
      READ(99,FARMAT) (ENTRYS(KK),KK=1,FULWRD)
6822  INITAL=FINAL+2
      IF(LPATH.EQ.0)  INITAL=INITAL+1
C     THIS WILL TAKE CARE OF EXTRA PARENTHSIS
      CALL  SEARCH(COMMA,NUMBER,10,BRNCH1)
      INITAL=FINAL+2
      CALL  SEARCH(COMMA,NUMBER,10,BRNCH2)
      INITAL=FINAL+2
      CALL  SEARCH(BLANK,NUMBER,10,BRNCH3)
      GO TO 652
68221 JPATH=2
C     ) IS A PART OF ARITH. EXPRESSION
```

```
      FINAL=KFINAL
      KMAX=(LENGTH/6)+1
      GO TO 5760
6830  INPLS3=I+3
      INITAL=ISAVE
C     WILL BE HELPFUL IN CASES WHERE - SIGN IS A PART OF
C     SUBSCRIPT, BUT ACTUAL OERATOR IS RELATIONAL
      IF(BUFFER(I+1).EQ.ALPBET(1))  INPLS3=INPLS3+1
      WRITE(99,300)(BUFFER(K),K=I,INPLS3)
      READ(99,304) OPRATR
      IF(BUFFER(I+1).EQ.ALPBET(1))  READ(99,305) OPRATR
      CALL PUTBRN(OPRATR,FOLLOW,BRNCH1,BRNCH2,BRNCH3,NN)
      IJKPRE=INITAL
      IJKFIN=FINAL
      INITAL=INPLS3+1
      J=INITAL
      UPLIMT=28
C     IT MAY NOT BE NEEDED ACTUALLU
      IF(FOUND.EQ.100)  GOTO 6831
      GOTO(781,782,480),FOUND
6831  CALL SEQUNC(LONGEX,M,JPREV)
      IF(TYPE.EQ.4)  ENTRYS(6)=OPRATR
      IF(LONGEX(1).EQ.RIGHTP.OR.LONGEX(1).EQ.PERIOD.AND.LONGEX(2
     1).EQ.RIGHTP)  GOTO 47000
      IF(LONGEX(1).EQ.PERIOD.AND.LONGEX(2).EQ.PERIOD)  GOTO 4705
     /6
      IF(LONGEX(1).EQ.LEFTP.AND.LONGEX(2).EQ.RIGHTP)  GOTO 47047
      IF(LONGEX(1).EQ.LEFTP.AND.LONGEX(2).EQ.COMMA.AND. LONGEX(3
     /).EQ.
     1RIGHTP.OR.LONGEX(1).EQ.LEFTP.AND.LONGEX(2).EQ.COMMA.AND.LO
     /NGEX(3)
     2.EQ.COMMA.AND.LONGEX(4).EQ.RIGHTP)  GOTO 47089
      GOTO 480
47000 ITRM2S=0
      IROUTE=1
      IQUIT=100
      LOGIFO=.TRUE.
      CALL SEARCH(RIGHTP,ALNUM,36,TERM2)
      GOTO 6834
47056 ITRM2S=0
      IROUTE=2
      LOGIFO=.FALSE.
      IQUIT=0
      CALL SEARCH(PERIOD,ALNUM,36,TERM2)
C     WE MUST MAKE SURE IF WE HAVE ENCOUNTERD A DECIMAL POINT OR
C     A FULL STOP
      IF(BUFFER(FINAL+2).NE.ALPBET(1).AND.BUFFER(FINAL+2).NE.ALP
     1BET(15))  GOTO 47059
      GOTO 6834
47059 ISAVE=INITAL
      INITAL=FINAL+2
```

```
      CALL SEARCH(PERIOD,NUMBER,10,TERM2)
      JSTMNT=       47059
      INITAL=ISAVE
      GOTO 6834
47047 IF(LONGEX(3).EQ.RIGHTP)  GOTO 47040
C     OTHERWISE IT MUST BE A .
      IF(LONGEX(3).NE.PERIOD)  GOTO 480
      ITRM2S=100
      LOGIFO=.FALSE.
      IQUIT=0
      IROUTE=4
      CALL SEARCH(PERIOD,ALNUM,36,TERM2)
      GOTO 6834
47040 ITRM2S=100
      IROUTE=3
      LOGIFO=.TRUE.
      CALL SEARCH(RIGHTP,ALNUM,36,TERM2)
      GOTO 6834
47089 CALL SEARCH(RIGHTP,ALNUM,36,TERM2)
      ISAVE=FINAL+2
      IF(BUFFER(ISAVE).EQ.RIGHTP)  GOTO 47085
      IF(BUFFER(INITAL).EQ.RIGHTP)  GOTO 47085
C     OTHERWISE IT MUST BE A .
      ITRM2S=100
      LOGIFO=.FALSE.
      IQUIT=0
      IROUTE=6
      GOTO 6834
C
C
403   UPLIMT=JPREV
      CALL GETENT(LONGEX,MAXNOP)
      GOTO 5021
C
C
47085 ITRM2S=100
      IROUTE=5
      LOGIFO=.TRUE.
      COTO 6834
6834  IMIN1=I-1
      IF(ITRM2S.EQ.100) FINAL=FINAL+1
      WRITE(99,10) (BUFFER(K),K=IJKPRE,IMIN1),MINUS,(BUFFER(K),K
     1=INITAL,FINAL)
      K=FINAL-IJKPRE-4+1+1
C     4 BECAUSE .OP. IS DELETED,+1 BECAUSE - IS INSERTED,AND+1 I
C    /S NORMAL
      FULWRD=K/6
      REMAIN=K-FULWRD*6
      CALL  BINBCD(FULWRD,FARMAT(2))
      CALL  BINBCD(REMAIN,FARMAT(4))
```

```
      IF(REMAIN.NE.0)  FULWRD=FULWRD+1
      IF(FULWRD.GT.5)  GOTO 480
      READ(99,FARMAT) (ENTRYS(K),K=1,FULWRD)
      READ(99,FARMAT) (ENTRY(LLL,K),K=1,FULWRD)
      IFULP1=FULWRD+1
      DO 6835 K=IFULP1,5
      ENTRYS(K)=BLANKS
6835  ENTRY(LLL,K)=BLANKS
C     NOT SURE WHETHER TO BE USED IN MAIN DT OR SUB DT. SO ENTER
C     AT BOTH PLACES
      INITAL=FINAL+1
      ASSIGN 403 TO LOGIF
      EXENT(LLL,RULE)=OPRATR
      ISTMN=6834
      JSTMNT=6834
      GOTO 62905
6830  LPATH=100
      GO TO 6805
725   GO TO 630
C
C***********************************************************************
C     HANDLES CONTINUATION CARDS OF IFS
C***********************************************************************
750   FOUND=100
      NOCONT=NOCONT+1
751   READ 10,COLS
      PRINT 110,COLS
      WRITE(0,12) N1,COLS
      IF(COLS(1).EQ.C)  GOTO 751
C     CHECK IF A COMMENT CARD
      ICONT=ICONT+1
      CALL  SQUEZE(COLS,JPREV)
      UPLIMT=72+NOCONT*66
      IUPTO =INITAL+UPLIMT-1
      CALL GETENT(LONGEX,MAXNOP)
      IF(FOUND.EQ.0)  GO TO 750
      IJKPRE=FINAL
      IJKFIN=FINAL
      INITAL=FINAL+2
      GOTO 50100
C     SET FOUND=100 SO THAT JPREV MAY BE SET TO PROPER IN
C     SQUEZE
C
C
780   ENTRYS(1)=TERM1
      ENTRYS(2)=DASH
      IF(TYPE.EQ.3) ENTRYS(2)=BLANKS
      ENTRYS(3)=TERM2
      ENTRY(LLL,1)=TERM1
```

```
      ENTRY(LLL,2)=DASH
      ENTRY(LLL,3)=TERM2
      IF(TYPE.EQ.3)  ENTRY(LLL,2)         =BLANKS
      ENTRY(LLL,4)=BLANKS
      ENTRY(LLL,5)=BLANKS
      ENTRYS(4)=BLANKS
      ENTRYS(5)=BLANKS
      EXENT(LLL,RULE)=OPRATR
      IPATH=1
      ASSIGN  503 TO LOGIF
      GOTO 62906
781   JSTMNT=781
C     THIS PATH FOR AND OPERATOR
      ENTRY(LLL,1)=TERM1
      ENTRYS(1)=TERM1
      ENTRYS(2)=BLANKS
      ENTRY(LLL,2)=BLANKS
78160 ENTRY(LLL,3)=BLANKS
      ENTRY(LLL,4)=BLANKS
      ENTRY(LLL,5)=BLANKS
      ENTRYS(3)=BLANKS
      ENTRYS(4)=BLANKS
      ENTRYS(5)=BLANKS
      EXENT(LLL,RULE)=T
      IPATH=4
      ASSIGN 503 TO LOGIF
      ISTMN=78128
      GOTO 62906
C*****************************************************************
782   JSTMNT=782
C     THIS IS FOR OR OPERATOR
      GOTO 78150
C*
C*
C*                                  WITH *NOT* OPRATR
783   ENTRY(LLL,1)=NOT
      ENTRYS(1)=NOT
C     FEB  1  9  6  9
C     THIS PATH FOR NOT OPERATOR
      ENTRYS(2)=TERM1
      ENTRY(LLL,2)=TERM1
      ENTRYS(2)=ENTRY(LLL,2)
      JSTMNT=738
      GOTO 78160
78850 CALL  SEARCH(PERIOD,ALNUM,36,ENTRY(LLL,2))
      IF(FOUND.EQ.0)  GOTO 7850
C     FAILURE MEANS ,INSTEAD OF . WE SHALL ENCOUNTER )
7850  CALL  SEARCH(RIGHTP,ALNUM,36,ENTRY(LLL,2))
      ENTRYS(2)=ENTRY(LLL,2)
      IF(FINAL.NE.IJKPRE)  GOTO 480
C     ASSUMING  THAT IF . IS NOT FOUND,) IS DEFINITELY FOUND
C    /TING
```

```
C       WITH () ARE NOT CONSIDERED
        LOGIFO=.TRUE.
        JSTMNT=7850
        GOTO 78160
790     UPLIMT=6
        INITAL=15
C       FEB  1  9  6  9
        GOTO 788
792     BRNCH1=STARS
        BRNCH2=STARS
        BRNCH3=STARS
C---------------   --------  ----------------+---    -  -------  ---
C
C       THE TRUE CONDITION OF LOGICAL IF HAS BEEN COMBINED WITH
C       .EQ.0 CASE OF ARITHMATIC IFS
C
C---------------  -  --------  ---------------- --    -  -------  ---
        IF(ORDER.EQ.GOTO4) BRNCH2=FOLLOW
        IF(ORDER.EQ.STOP4.OR.ORDER.EQ.RETURN.OR.ORDER.EQ.CALL.AND
       1FOLLOW.EQ.I38888)  BRNCH2=FOLLOW
        FOUND=3
        GOTO652
C**************************************************************************
C ****.********.****************************************************
C *
C *                 PROCESSING FOR GOTO AND COMPUTED GOTO
C **************************************************************************
C       IT IS EXPECTED THAT NOBODY WILL ISE A CONTINUATION CARD
C       FOR GOTO STATEMENT, BUT ONE MAY USE IT FOR COMPUTED AND
C       ASSIGNED GOTO.
C       SEE 370 FOR CONTINUATION CARDS
800     INITAL=11
        CALL  SEARCH(EQUALS,ALNUM,36,GOAL)
        IF(FOUND.EQ.100)  GO TO 310
        IF(BUFFER(11).EQ.LEFTP)  GO TO 850
        CALL SEARCH(COMMA,ALNUM,36,INDEX)
        IF(FOUND.EQ.0)  GOTO 8000
        INITAL=FINAL+3
        IPATH=2
C       THIS PATH FOR ASSIGNED GOTO
        GO TO 854
C       THIS PATH FOR GOTO

        IF(FOUND.EQ.0)  GO TO 480
```

```
      BRNCH1=GOAL
      BRNCH2=GOAL
      BRNCH3=GOAL
      ENTRYS(1)=GOTO4
      GOTO 652
C ******************************************************************
C *                                    COMPUTED GO TO
C ******************************************************************
850   INITAL=12
      IPATH=1
C     THIS PATH FOR COMPUTED GOTO
854   JK=1
855   CALL  SEARCH(COMMA,NUMBER,10,LIST(1,JK))
      IF(FOUND.EQ.0)  GO TO 870
      INITAL=FINAL+2
      JK=JK+1
      GO TO 855
C     GOBACK TO FIND THE NEXRT  BRANCH
870   CALL  SEARCH(RIGHTP,NUMBER,10,LIST(1,JK))
      INITAL=FINAL+3
C     .......F),I   HENCE FINAL+3
C     CAN INSERT FOR CONTINUATION CARDS OF COMPUTED AND ASSIGN
C    / GOTO
      IJK=JK
      IF(IPATH.EQ.2)  GOTO 880
      DO 8701 K=1,IJK
8701  EXENT(1,K)=BCDTAB(K)
C     BCDTAB CONTAINS THE LEFT JUSTIFIED SERIAL NOS. IN BCD
      CALL  SEARCH(BLANK,ALNUM,36,INDEX)
8700  ENTRY(1,1)=INDEX
      ENTRY(1,2)=EQUALS
      ENTRY(1,3)=BLANKS
      ENTRY(1,4)=BLANKS
      ENTPY(1,5)=BLANKS
      IPATH=2
C     IPATH HAS SERVED ITS PURPOSE.NEW VALUES CAN BE ASSIGNED TO
C    / IT FOR
C     FURTHER USE  AT 62992 ETC
      RULE=IJK
      IRULES=IJK
      NOSTRO=1
      LLL=1
      SUBTAB=SUBTAB+1
      ITEMP=DASH
      BRNCH2=GOTO4
      FOLLOW=GOTO4
C     THIS IS NOT PERTINENT. USED FOR PROPER ROUTING
C     ROUTING AT 653
      GO TO 62965
```

```
880   DO 882 I=1,IJK
882   EXENT(1,I)=LIST(1,I)
C
C
C ************************************************************************
C *                                STOP
C
C                                  CALL EXIT
C*************************************************************************
900   ENTRYS(1)=STOP4
920   BRNCH1=DASH
      BRNCH2=DASH
      BRNCH3=DASH
      GO TO 9990
940   ENTRYS(1)=RETURN
      GO TO 920
960   WRITE(99,300)(BUFFER(K),K=11,14)
      READ(99,304) FOLLOW
      IF(FOLLOW.EQ.EXITT)  GOTO 970
      GOTO 310
970   ENTRYS(3)=CALL
      ENTRYS(4)=FOLLOW
      GO TO 900
9990  GO TO 652
C
C
C*************************************************************************
C ************************************************************************
C*
C*
C                                  THIS IS THE END OF CLASFY
C*
C*
C ************************************************************************
99904 CALL  PARSEM
C     THERE IS A NORMAL STOP IN PARSEM (PARSE MAIN).HOWEVER, FOR
C     THE SAKE OF COMPLETENESS KEEPING A STOP HERE
29999 IF(COLS(30).EQ.ALPBET(16))  GOTO 99904
C     THIS IS THE CASE OF PARSE
C*************************************************************************
      IF(ISSTOP.EQ.100)  GOTO 5599
31111 PAUSE  11111
C     THIS IS FOR PAPER MOUNTING TO GET OUTPUT IN A NICE FORMAT
      N1SAVE=N1
      N1=N1-1+ICONT-1
      NOSUBT=SUBTAB
C*************************************************************************
C     THIS IS TO HAVE AN IDEA OF TOTAL CARDS TO BE PRINTED
      REWIND  0
```

```
      REWIND  1
      REWIND  2
      REWIND 4
 2222 FORMAT(1H ,36A1,A6,2X,8(5X,A5))
      IF(COLS(25).EQ.ALPBET(2))  GOTO 55502
      TABLE=ALPBET(2)
      PRINT 215,TABLE
      DO 5550 JA=1,NOTAB2
      READ (2,319) (BUFFER(I),I=7,42),NN,STMNNO,BRNCH1,BRNCH2,BR
     /NCH3
 5550 PRINT2222, (BUFFER(I),I=7,42),NN,STMNNO,BRNCH1,BRNCH2,BRNC
     /H3
55502 IF(COLS(15).EQ.ALPBET(1))  GOTO 29996
      TABLE=ALPBET(1)
      NCRD=ROWTA1
      IF(NCRD.EQ.0)  CALL EXIT
C     THIS IS THE CASE WHEN TABLE A IS EMPTY,I.E. NO CONTROL
C     STATEMENT,WHENCE NO D.T.
      PRINT 21501
21501 FORMAT(1H1,*NOTE. IN THE FOLLOWING TABLE,FOR THE CONDITION
     /S CONTAI
     1NING DECLARED LOGICAL VARIABLESTHE TRUE CONDITION */1H ,*-
     /---*,3X,
     2*HAS BEEN COMBINED WITH EQZ CASE OF ARITHMATIC EXPRESSIONS
     /.*///)
      PRINT 215,TABLE
      DO  29995 J=1,NCRD
      READ(1,320) (ENTRY(J,KK),KK=1,7),N(J),STMNT(J),BRNCH(J,1),
     /BRNCH(J,
     12),BRNCH(J,3)
      PRINT 22001,(ENTRY(J,M),M=1,7),N(J),J,STMNT(J),(BRNCH(J,M)
     /,M=1,3)
22001 FORMAT(1H ,7A6,A6,I3,8(5X,A5))
29995 CONTINUE
29996 IF(NOSUBT.EQ.0)  GOTO 2992
      DO 29993 I=1,NOSUBT
      READ(4,62985) HEADER(I),NOSTRO,IRULES
      PRINT 20104
      PRINT 62981,HEADER(I),NOSTRO,IRULES
      PRINT 20104
62981 FORMAT(1H ,*SUBTABLE  *,A5,I4,I4)
      PRINT 26001,(NUMB(JKL),JKL=1,IRULES)
      DO 29991 K=1,NOSTRO
      READ(4,62987)(ENTRYS( IJ),IJ=1,5),(EXENT(K,JR),JR=1,IRULES
     /)
29991 PRINT 12346, (ENTRYS( IJ),IJ=1,5),(EXENT(K,JR),JR=1,IRULES
     /)
      PRINT 42344
      PRINT 42344
42344 FORMAT(1H ,*---------- ----------------  -- ----------
     /-------
     1-----------------------------------*)
```

```
      READ(4,62989) (LIST(1,JR),JR=1,IRULES)
29993 PRINT 26005,  (LIST(1,JR),JR=1,IRULES)
2992  NOMERG=0
      IF(CARYON.EQ.0.0)  CALL EXIT
C
C
C
215   FORMAT(//////56X,*-------------------------*//56X,*  T A B
     /L E
     1  *,A2,*  * //56X,*-------------------------*////1X,*-------
     /--------
     2----------  - -------- -------------- ---   - ------- ---
     /------
     3----------- - -------- ------------------ --*/21X,
     4*ENTRY*,17X,*N*,12X,*STMNT*,3X,*  BRNCH1    BRNCH2  *, *
     /BRNCH3
     5*,*CAME FROM*,*   LT 0   *,*   EQ 0   *,
     /*   GT 0
     6*/1X,*--------------- ----------------  -- ------- ---
     /--------
     7----------    -------- -------------- ---   - ------- ---
     /---*//)
      DO 10000 IJK=1,3
      DO 10000 J=1,100
10000 EXCUTD(J,IJK)=DASH
      DO  20011  J=1,40
      DO 20011 JK=1,32
20011 LIST(J,JK)=BLANKS
      DO 99991 J=1,100
      MASTER(J)=DASH
99991 PREVNO(J)=DASH
11112 FORMAT(7A6,2X,A6,4(2X,A5))
C
C
C*****************************************************************
C
C
C     ACTUAL 2ND PHASE STARTS FROM HERE
C
C
C*****************************************************************
      DO 2001 ICON=1,40
      DO 2001 RULE =1,40
2001  EXENT(ICON,RULE)=BLANKS
      PREVNO(1)=BEGIN
      MASTER(1)=BEGIN
      IF(STMNT(1).EQ.BLANKS)  STMNT(1)=BEGIN
C
C
```

```
      RESET=0
      EXINT=1
      LOOP=0
C     POINTER FOR SEEING IF LOOPING WAS EVER THERE
      RULE=1
      IRULES=0
      ENDING=0
C     PAUSE 12345
C      THIS IS TO SET SENSE SWITCJ
      SUBDT=.FALSE.
C     SUBDT IS FALSE TILL PARSING IS COMPLETE.IT IS TRUE WHILE P
C    /ROCESS-
C     ING THE COLS OF GOTOS
      L=1
      J=1
      LFF=1
      FILLED=0
      LF=1
C
      RENTER(1)=1
C     1ST CONDITION IS ALWAYS ENTERED SEQUENTIALLY OR OTHERWISE
      IPATH=1
C
2010  MCAME=BCDTAB(J)
C
C
C     MCAME IS LATER USED TO HAVE PREVNO(JJ) AND RETFRM.THIS
C     CONTAINS BCD MODE VALUE OF ..J.. BUT HAS NOTHING TO DO WITH
C     N1= SOURCE STATEMENT  OF PHASE1
C
C
      PRINT 20104
20104 FORMAT(///)
      IF(ENTRY(J,1).NE.STOP.AND.ENTRY(J,1).NE.RETURN.AND.ENTRY(J
     1,1).NE.GOTO)   GOTO 26208
20105 K=J
      IF(ENTRY(J,1).EQ.GOTO)   EXCUTD(J,3)=DASH
217   IF(EXINT.EQ.10.AND.J.EQ.RESET)   GO TO 21811
      I=1
      IF(EXCUTD(J,I).EQ.DASH)    GO TO 2020
      IF(SUBDT.AND.J.EQ.NCRD)   GOTO 46204
C     THIS IS PERTINENT ONLY WHILE PROCESSING SUB DTS OF GOTOS
      IF(J.EQ.1)   GOTO 2192
2171  I=2
      IF(EXCUTD(J,I).EQ.DASH)    GO TO 2020
      I=3
      IF(EXCUTD(J,I).EQ.DASH)    GO TO 2020
C
C     THIS IS THE CASE IF WE ARE RETURNING BACK AFTER STOP.
      IF(J.NE.1)    GO TO 218
```

```
21710 PRINT 21711
C     THIS WILL ALSO BE APPROACHED IN THE CASE OF A DT HAVING
C     MORE THAN  THE ANTICIPATED NO. OF RULES OR CONDITIONS
      PRINT 21712
      PRINT 21713
21711 FORMAT(1H1,51X,28(1H-)//52X,*D E C E S I O N    T A B L E
     /*//1H ,
     151X,28(1H-)///)
21712 FORMAT(1H ,131(1H.))
21713 FORMAT(/)
      PRINT 26001,(NUMB(JKL),JKL=1,16)
      PRINT 21713
      PRINT 21712
      ITEMP1=IRULES
      ITEMP=LMIN1
      IF(ITEMP1.GT.16)  IRULES=16
      DO 22347  LLL=1,LMIN1
22347 PRINT 12346,(CONDSN(LLL,IJK),IJK=1,5),(EXENT(LLL,JRULE),JR
     1ULE=1,IRULES)
      DO 26204 LMN=1,LFF
26204 PRINT 26005,(LIST(LMN,JRULE),JRULE=1,IRULES)
      IF(ITEMP1.LE.16)  GOTO 26207
      IRULES=ITEMP1
      DO  26226  JKL=1,16
26226 NUMB(JKL)=JKL+16
      PRINT 26001,(NUMB(JKL),JKL=1,16)
      DO 22348 LLL=1,LMIN1
22348 PRINT 12346,(CONDSN(LLL,IJK),IJK=1,5),(EXENT(LLL,JRULE),JR
     1ULE=17,IRULES)
      DO 26206  LMN=1,LFF
26206 PRINT 26005,(LIST(LMN,JRULE),JRULE=17,IRULES)
26207 CONTINUE
C     INSERT CALL PARSE TWICE          OTOS
C     WRITE ON TAPE THE ALREADY OBTAINED DT
      IF(FILLED.GT.0)  GOTO 26201
      STOP
C--------------  - -------- ----------------  -- ------- ---
C
C     PROCESSING OF S U B  T A B L E S  OF  G O T O S
C
C--------------  - -------- ----------------  - ------- ---
26201 SUBDT=.TRUE.
      NCRD=NCRD+1
C     CREATING A NEW ROW IN TABLE A
      ICOL=0
      J=NCRD
      DO 46201 I=2,7
46201 ENTRY(J,I)=BLANKS
      ENTRY(J,1)=GOTO4
      N(J)=BLANKS
```

```
C
C
46204 ICOL=ICOL+1
      IF(ICOL.GT.FILLED)  STOP
      DO 46205 I=1,IRULES
      DO 46205 K=1,L
46205 EXENT(K,I)=BLANKS
      DO 46206 IJK=1,3
      DO 46206 JK=1,NCRD
46206 EXCUTD(JK,IJK)=DASH
      DO 46207 J=1,LFF
      DO 46207 JK=1,IRULES
46207 LIST(J,JK)=BLANKS
      DO 46208 J=1,NCRD
      MASTER(J)=DASH
46208 PREVNO(J)=DASH
      DO 46209 K=1,L
      DO 46209 KK=1,7
46209 CONDSN(K,KK)=BLANKS
C     FINALLY YOU MAY HAVE TO KEEP A SEPARATE PRINT FOR SUCH TAB
C    /LES
      DO 46210 I=1,NCRD
46210 RENTER(I)=0
      RENTER(NCRD)=1
      J=NCRD
C     NCRD TH ROW OF TABLE A(NEWLY CREATED) SHALL ALWAYS HAVE AN
C    / ELEMENT
C     OF COLS
      PREVNO(J)=BEGIN
      MASTER(J)=BEGIN
      STMNT(J)=BEGIN
      CALL BINBCD(NCRD,NN)
      CALL REMLBL(NN,NUMBER(10))
      BCDTAB(NCRD)=NN
      N(J)=BCDTAB(J)
      RESFT=0
      EXINT=1
      ENDING=0
      BRNCH(J,1)=COLS(ICOL)
      BRNCH(J,2)=DASH
      BRNCH(J,3)=DASH
      RULE=1
      IRULES=1
      L=1
      LF=1
      LFF=1
C     FINALLY WHEN PRINTS ARE SUPRESSED ,IT WILL PROVIDE  THE PR
C    /OPER
C     SUB TITLE
```

```
      I=1
C     ULTIMATELY WE COULD KEEP .. GOTO 2020 OR STILL BETTER ..GO
C    /TO 2024
C     INSTEAD OF  GOTO 2010
      GO TO 2010
C
C
26208 CALL RIGHTJ(N(J),N1J1,BLANK,NUMBER(10))
      CALL BCDBIN(N1J1,NJ)
      ISTMN=26208
      RENTER(J)=1
      GOTO 20105
219   DO 2191 K=1,3
2191  EXCUTD(J,K)=DASH
      JJ=J
      IPATH=100
      GOTO 250
2192  DO  28501 IJ=2,NCRD
28501 RENTER(IJ)=0
      GOTO 2171
218   DO 2181  K=1,3
2181  EXCUTD(J,K)=DASH
      IPATH=1
C     IF IPATH IS NOT RESET TO 1 ,WE SHALL GET A SUPERFLUOUS RULE
C     IN CERTAIN CASES IF LAST ROUTE HAD BEEN 2026,28500,250 ETC
      EXINT=10
      RESET=J
C     EXINT IS FOR INITIALISING ECXCUTED
      IF(MASTER(J).NE.PREVNO(J).AND.LOOP.EQ.1)  GOTO 21812
C     INSERTED APRIL 28,1969. SHOULD HELP IN GETTING PROPER EXIT
C     EXAMPLE-  GAUSS SEIDAL METHOD WITH EXTRA ARTIFICIAL CONDIT
C    /ION
C
251   JJ=J
C     POINTER FOR RULE  THIS WILL BE TESTED BEFORE INCREMENTING
C    /RULE
      GO TO 250
C
C     SO THAT LOGIC MAY BE CLEAR DELEBRATELY KEEPING IT AS A GO
C    /TO
21812 RETFRM=MASTER(J)
      NPATH=21812
      GOTO 25000
21811 EXINT=0
      I=1
C     IF I IS NOT SET TO 1 THEN LAST VALUE OF I WILL BE TAKEN AN
C    /D ORDER
C     OF RULES GETS ALTERED
```

```
C
2020  MBRNCH=BRNCH(J,I)
      EXCUTD(J,I)=BRNCH(J,I)
      IF(ENTRY(J,1).EQ.GOTO.OR.ENTRY(J,1).EQ.STOP)  GOTO 20240
      IF(ENTRY(J,1).EQ.LOGDT.AND.I.EQ.2)  GOTO 279
      IF(ENTRY(J,1).EQ.LOGDT)  GOTO 20272
      IF(ENTRY(J,1).EQ.RETURN)  GOTO 2024
C
C     ENTPY(J,1) CAN NEVER BE A STOP BECAUSE OF THE PROGRAM SETU
C    /P
C     ***********************************************************
C           PROCESSING FOR IF STARTS
C
C     ***********************************************************
C
C     THIS IS TO TEST FRON CONDITIONS BECOMING DUPLICATE
C
C     ***********************************************************
C     L IS TO BE SAVED BECAUSE IT IS THE LATEST VALUE
2021  LFINAL=L
      IF(L.LT.2)  GO TO 23500
      LMIN1=L-1
      DO 2350  LJK=1,LMIN1
      DO  23501  IJK=1,7
      IF(ENTRY(J,IJK).NE.CONDSN(LJK,IJK))  GO TO 2350
23501 CONTINUE
C     A MATCHING CONDITION WHICH WAS ALREADY THERE HAS BEEN FOUN
C    /D
      L=LJK
      IPATH=10
C     THIS IS JUST TO TAKE CARE OF TWO BRANCHES OF A TREE HAVING
C    / A COMMO
C     N  SUB SEXCTION
      GO TO 20260
2350  CONTINUE
C     IF A NORMAL EXIT  IT IS A NEW CONDITION
23500 IPATH=0
C     THIS IS JUST TO TAKE CARE OF TWO BRANCHES OF A TREE HAVING
C    / A COMMO
C     N  SUB SEXCTION
      DO 2351  IJK=1,7
2351  CONDSN(L,IJK)=ENTRY( J,IJK)
      LJK=L
20260 IF(BRNCH(J,1).EQ.BRNCH(J,2).AND.BRNCH(J,1).EQ.BRNCH(J,3).A
     /ND.BRNCH
     1(J,1).EQ.STARS)  GOTO 20264
      IF(BRNCH(J,1).EQ.BRNCH(J,2).AND.I.LE.2)  GOTO 20261
      IF(BRNCH(J,1).EQ.BRNCH(J,3).AND.I.NE.2) GO TO 20262
```

```
      IF(BRNCH(J,2).EQ.BRNCH(J,3).AND.I.GE.2) GO TO 20263
      GO TO (231,232,233),I
231   EXENT(L,RULE)=LTZ
      GO TO 235
232   EXENT(L,RULE)=EQZ
      GO TO 235
233   EXENT(L,RULE)=GTZ
235   IF(IPATH.EQ.10)  L=LFINAL-1
      L=L+1
C     LATER WHEN WE INCREMENT L IT WILL BE OK
C     IF IT IS NOT 10 IT MUST BE 0 AND L HAS NOT BEEN TEMPERED W
C    /ITH
      PRINT 907,(CONDSN(LJK,IJK),IJK=1,7),MBRNCH,EXENT(LJK,RULE)
907   FORMAT(1H ,7A6,*FOR *,A5,* HAS EXTENDED ENTRY AS  *,A5)
C****************************************************************
C     WE ARE NOW SEARCHING FOR A STATEMENT WHICH MATCHES THE
C****************************************************************
C     BRANCH
20240 IF(MBRNCH.EQ.STARS)  GO TO 2090
2024  CONTINUE
      DO 2025 JJ=1,NCRD
      IF(STMNT(JJ).EQ.MBRNCH)  GO TO 2026
2025  CONTINUE
      PRINT 2027,MBRNCH
2027  FORMAT(1H ,*DATA FOR PHASE2 SEEMS TO BE WRONG*,A6,*IS MISS
     /ING IN S
     1TMNT COLUMN*/58X,*OR*/1H ,*IS TO BE OBTAINED FROM TABLE 2*
     /)
C
C****************************************************************
      REWIND  2
      DO 20271 JA=1,NOTAB2
      READ (2,319) (BUFFER(I),I=7,42),NN,STMNNO,BRNCH1,BRNCH2,BR
     /NCH3
      IF(STMNNO.EQ.MBRNCH)  GOTO 2400
20271 CONTINUE
C     THIS SHOWS THAT IN  AN EFFECTIVE  GOTO  A DECLERATION HAS
C     BEEN MET WITH. I B M FORTRAN IV DOES NOT POINT OUT AN
C     ERROR MESSAGE  FOR
C               GOTO  10
C          10    FORMAT(1H ,............)
C     BUT WATFOR  DOES. SINCE ALL DECLERATIONS HAVE BEEN THROWN
C     OUT AND ARE NOT IN TABLES A AND B, IT IS A MISTAKE OF THE
C     PROGRAMMER.
C     SUITABLE MESSAGE IS PRINTED OUT
C
      PRINT 2027,MBRNCH
      ISTMN=20271
```

```
      PRINT 24002,ISTMN
      STOP
20272 IF(I.EQ.1)  EXCUTD(J,3)=BRNCH(J,3)
      GOTO 20240
2026  PREVNO(JJ)=MCAME
      IPATH=1
      IF(MASTER(JJ).EQ.DASH)  MASTER(JJ)=PREVNO(JJ)
C     THIS WILL BE USEFUL WHEN IT STARTS LOOPING IN ..DTENT(..,P
C    /REVNO)..
      K=LJK
C     THIS IS NOT NEEDED IN THE FINAL SHAPE
C     IF COMING TO 2026 VIA 2400 CONDITION LISTED WILL BE THE
C     ACTUALLY THE PREVIOUS ONE. IT DOES NOT MATTER IN FINAL
C     ANALYSIS
C
C
      IF(ENTRY(JJ,1).EQ.STOP)  GO TO 250
      IF(ENTRY(JJ,1).EQ.RETURN)          GOTO 250
      IF(ENTRY(JJ,1).EQ.GOTO)  GO TO 270
      IF(RENTER(JJ).EQ.1)  GOTO 28500
43109 ISTMN=2026
      IF(ENTRY(JJ,1).EQ.LOGDT.AND.BRNCH(JJ,1).EQ.DASH)  GOTO 280
C     THUS ROUTING TO 280 WILL BE DONE ONLY FOR SUBDTS OF ..GOTO
C     FOR SUBDTS OF ..IFS.. ROUTING IS THROUGH 2020,279,280...
C
C
C     J IS THE RECORD WITH WHICH WE START  AND JJ IS THE ONE FOR
C     WHICH MATCH IS FOUND
C
C
      IF(ENTRY(J,6).EQ.CALL.AND.MBRNCH.NE.STARS)  GOTO 24098
240   CONTINUE
C     KEPT INSTEAD OF 240 CONTINUE AND LF=LF+1
C     L STANDS FOR CONDITION NO. LF FOR ACTION NO. AND LL FOR CU
C    /RRENT
C     VALUE OF LIST
      J=JJ
      IF(L.GT.40.OR.IRULES.GT.40.OR.LFF.GT.40)  GOTO 245
      GO TO 2010
24098 LIST(LF,RULE)=ENTRY(J,7)
      LF=LF+1
      GOTO 240
2400  CONTINUE
      LIST(LF,RULE)=MBRNCH
      LF=LF+1
      CALL RIGHTJ(NN,NN,BLANK,NUMBER(10))
      CALL BCDBIN(NN,NNBIN)
C     KEEPING 2,NCRD SO THAT $BEGIN$ IS NOT CONVERTED BINARY
      DO 24001 JJ=1,NCRD
C     CHANGED LOOP FROM JJ=2,NCRD TO JJ=1,NCRD APRIL 3,1969
```

```
      CALL RIGHTJ(N(JJ),N1 J1,BLANK,NUMBER(10))
      CALL BCDBIN(N1J1,NJJ)
      IF(NJJ.GT.NNBIN)  GO TO 24010
C     FINALLY KEEP 2026 INSTEAD OF 14010
C     NO. SIR. YOU CAN NOT      APRIL 18,1969
24001 CONTINUE
      PAUSE 33333
      ISTMN=24001
      PRINT 24002,ISTMN
24002 FORMAT(/1H ,*STOPING AFTER PAUSE AT *,2X,I7/)
      STOP
24010 IF(ENTRY(JJ,1).EQ.LOGDT)  MBRNCH=STMNT(JJ)
      GOTO 2026
24012 MBRNCH=STMNT(JJ)
      GOTO 2026
C
C
C
20261 EXENT(L,RULE)   =LEZ
      EXCUTD(J,2)=BRNCH(J,1)
      GO TO 235
20262 EXENT(L,RULE)=NEZ
      EXCUTD(J,3)=BRNCH(J,1)
      GO TO 235
20263 EXENT(L,RULE)=GEZ
      EXCUTD(J,3)=BRNCH(J,2)
      GO TO 235
20264 EXENT(L,RULE)=ENTRY(J,6)
C     THIS IS BEING INSERTED BUT IN CERTAIN CASES IT WILL BE
C     MODIFIED AT VARIOUS STAGES
      WRITE(99,305)  ENTRY(J,6)
      READ(99,300)  (BUFFER(IK),IK=1,5)
      IF(BUFFER(2).EQ.ALPBET(12).AND.BUFFER(3).EQ.ALPBET(20)) IO
     /P=1
      IF(BUFFER(2).EQ.ALPBET(12).AND.BUFFER(3).EQ.ALPBET( 5)) IO
     /P=2
      IF(BUFFER(2).EQ.ALPBET( 5).AND.BUFFER(3).EQ.ALPBET(17)) IO
     /P=3
      IF(BUFFER(2).EQ.ALPBET( 7).AND.BUFFER(3).EQ.ALPBET( 5)) IO
     /P=4
      IF(BUFFER(2).EQ.ALPBET( 7).AND.BUFFER(3).EQ.ALPBET(20)) IO
     /P=5
      IF(BUFFER(2).EQ.ALPBET(14).AND.BUFFER(3).EQ.ALPBET( 5)) IO
     /P=6
      EXCUTD(J,I)=BRNCH(J,I)
C     BESIDES THE ABOVE CERTAIN OTHER EXCUTD MAY HAVE  TO BE
C     INSERTED AT VARIOUS PLACES
      GOTI  (201,202,203),I
201   GOTO(2045,2012,2013,2014,2015,2016),IOP
```

```
202    GOTO(20211,2022,2045,20241,20251,20269),IOP
203    GOTO(2031,2032,2033,2045,2045,2045),IOP
2012   EXCUTD(J,2)=BRNCH(J,2)
       GOTO 2045
2013   EXCUTD(J,3)=BRNCH(J,3)
       EXENT(L,RULE)=NEZ
       GOTO 2046
2014   EXENT(L,RULE)=NEZ
       GOTO 2046
2015   EXCUTD(J,2)=BRNCH(J,2)
       EXENT(L,RULE)=LEZ
       GOTO 2046
2016   EXCUTD(J,3)=BRNCH(J,3)
       GOTO 2045
20211  EXCUTD(J,3)=BRNCH(J,3)
       EXENT(L,RULE)=GEZ
       GOTO 2046
2022   EXCUTD(J,3)=BRNCH(J,3)
       GOTO 2045
20241  EXCUTD(J,3)=BRNCH(J,3)
       GOTO 2045
20251  EXENT(L,RULE)=LEZ
C      WE SHOULD NOT REACH THIS
       GOTO 2046
20269  EXENT(L,RULE)=EQZ
       GOTO 2046
2031   EXENT(L,RULE)=GEZ
C      WE SHOULD NOT REACH THIS
       GOTO 2046
2032   EXENT(L,RULE)=GTZ
       GOTO 2046
2033   EXENT(L,RULE)=NEZ
       GOTO 2046
C      WE SHOULD NOT REACH THIS
C      WE SHOULD NOT REACH THIS
2045   WRITE(99,62996) PLUS,N(J)
20451  READ(99,5)  LIST(LF,RULE)
       LF=LF+1
       GOTO 235
2046   WRITE(99,62996)  PLUS,N(J+1)
       GOTO 20451
C      THIS IS TO KEEP A REFERENCE MARK IN DT
245    PRINT 246,L,IRULES
246    FORMAT(1H ,*NO OF RULES OR NO OF CONDITIONS HAS EXCEEDED T
      /HE ANTIC
      1IPATED MAXIMUM*/,1H ,*NO OF CONDITIONS=*,I3,*  NO OF RULES
      /=*,I3///
      2/20X,*THE PARTIAL DECISION  TABLE IS GIVEN BELOW*///)
       GOTO 21710
C      PROCESSING FOR IF ENTRY IS OVER
```

```
C*****************************************************************************
C              PROCESSING FOR STOP STARTS
C
C*****************************************************************************
250    RETFRM=PREVNO(JJ)
       IF(ENTRY(JJ,1).EQ.STOP)  LIST(LF,RULE)=STOP
       IF(ENTRY(JJ,1).EQ.RETURN)  LIST(LF,RULE)=RETURN
       IF(IPATH.EQ.100)  LIST(LF,RULE)=SELF
       IF(LF.EQ.0)  GO TO 25000
C      INSTEAD  INSERTED ON 8868 FOR TESTING ITERATIVE LOOPS
C      AT THE PRESENT STMNT OF STOP IS INCLUDED IN LIST
C
C
C      THE LAST ELEMENT HAS BEEN BLANKED OUT BECAUSE STOP HAS
C      BEEN ENCOUNTERED
25000  IF(LF.LE.LFF)    GO TO 2910
       LFF=LF
C      THIS IS FOR FINAL VALUE OF LF
2910   CONTINUE
C      KEPT INSTEAD OF 2910 LF=LF-1
       LMIN1=L-1
       DO 2921  LMN=1,LMIN1
2921   EXENT(LMN,RULE+1)=EXENT(LMN,RULE)
       FINAL=1
       IF(RULE.GT.1) CALL DTENT(EXENT,RULE,RETFRM,NCRD,ENTRY,LMIN
      /1,LF,
      1CONDSN,PREVNO,LFF,LIST,STMNT,HEADER)
C      NOTE NOW PREVNO IS A LOCAL VARIABLE IN DTENT
       IF(RULE.GT.1.AND.FINAL.EQ.2) CALL DTENT(EXENT,RULE,RETFRM,
      /NCRD,
      1ENTRY,LMIN1,LF,CONDSN,MASTER,LFF,LIST,STMNT,HEADER)
C
       JPATH=1
293    DO 253  J3=1,NCRD
       IF(BCDTAB(J3).EQ.RETFRM)  GOTO 2531
253    CONTINUE
       PAUSE 66666
       ISTMN=253
       PRINT 24002,ISTMN
       STOP
2530   INITAL=BUFFER(600)
       FOUND=RULE-1
C      THE ABOVE TWO VARIABLES ARE USED TO ECONOMOISE IN SPACE
C      ONLY AND HAVE NOTHING TO DO WITH INITAL AND FOUND
       DO 25300 K=INITAL,FOUND
       DO 25300 KK=1,LF
25300  LIST(KK,K)=LIST(KK,RULE)
       GOTO 2600
C      INSERTED        MAY 7,1969
2531   IF(BUFFER(600).NE.RULE)  GOTO 2530
C
```

```
2600   IF(ENTRY(J3,1).EQ.GOTO)  GO TO 26006
       IRULES=RULE
C      ***************************************************************
       IF(ENTRY(JJ,1).EQ.STOP)  RULE=RULE+1
       IF(ENTRY(JJ,1).EQ.RETURN)  RULE=RULE+1
       IF(IPATH.EQ.100)  RULE=RULE+1
       IF(ENTRY(JJ,1).EQ.STOP.OR.ENTRY(JJ,1).EQ.RETURN.OR.IPATH.E
      1Q.100)  LF=1
26000 LMIN1=L-1
       IF(JPATH.NE.2)  J=J3
C      FOR JPATH=2,  J HAS ALREADY BEEN SET
26001 FORMAT(1H ,*ENTRY*,26X,16(*RULE*,I2))
12346 FORMAT(/1H ,5A6,1X,16A6/)
26005 FORMAT(1H ,*GO TO*,26X,16A6)
       GO TO 2010
C      ***************************************************************
C      PORTIONS OF LOGICAL IF TO BE INSERTED
2090   JPLUS1=J+1
       IF(ENTRY(JPLUS1,1).EQ.LOGDT)  MBRNCH=STMNT(JPLUS1)
       IF(N(JPLUS1).EQ.(N(J)+1))  GO TO 20912
C      IT CAN NEVER BE WITH N(J) AS IN BCDMODE
20912 JJ=JPLUS1
       GO TO 2026
C      ***************************************************************
C      PROCESSING FOR GOTO STARTS
270    DO  271  K=1,2
271    EXCUTD(JJ,K)=BRNCH(JJ,K)
C      SHALL HAVE TO LOOK INTO IT .ACTUALLY IF WE KEEP K=3 ALSO
C     /  IT IS
C      TAKEN AS IF RETUTRNING FROM STOP,AFTER TESTING FOR EXCUTED
       GO TO 240
C
C      PART PROCESSING OF GO TO ENTRY IS OVER RESR OF IT IS IN IF
C     / BLOCK
C      ***************************************************************
26006 RETFRM=PREVNO(J3)
       JSTMN=293
       J=J3
C      J=J3  WILL BE USED AT 26000
       IF(SUBDT.AND.RETFRM.EQ.BEGIN)  GOTO 26008
C      THIS WILL BE USEFUL FOR GETTING CONTINUATION OF D.T.S
       GO TO 293
26008 JPATH=2
       GOTO 26000
C      ***************************************************************
279    MBRNCH=STMNT(J)
280    NOMERG=NOMFRG+1
       LMIN1=L-1
       DO 284 K=1,20
```

```
      BUFFER(K)=BLANKS
284   LONGEX(K)=BLANKS
      DO 285 K=1,LMIN1
      BUFFER(K)=EXENT(K,RULE)
285   LONGEX(K)=EXENT(K,RULE)
      BUFFER(600)=RULE
C     SAVE THE CURRENT VALUE OF RULES. WILL BE USED FOR
C     ACTION SET
      CALL MERGE(NOMERG,MBRNCH,L-1,NXTBRN,FOUND,L,NRULES,LONGEX,
     1KRULES,J)
C     LONGEX IS USED TO ECONOMISE IN STORAGE. IT HAS SERVED ITS
C    /ROLE
C     IN PHASE 1
      L=L+1
      RULE=NRULES
C     L, THE 6 TH ARGUMENT IS THE TOTAL NO. OF UNIQUE CONDITIONS
C     MET IN MERGE AT RETURN.
C     SIMILARLY NRULES IS TOTAL NO. OF RULES
C     KRULES THE RETURNED VALUE IS USED IN CALLING UNIQUE
C     NOW THE CURRENT RULE UNDER CONSIDERATION IS
C                 RULE=NRULES=TRULES
C     OF MERGE
C
C
      GOTO  (28077,28060,28070),FOUND
C     FOUND=1  FOR SUB TABLE ALREADY ACCOUNTED
C     FOUND=2  FOR IFS
C     FOUND=3  FOR GOTOS
28060 MBRNCH=NXTBRN
      GOTO 20240
28065 CALL UNIQUE(LONGEX,COLS,FILLED,1)
C     1 BECAUSE ONLY 1 LONGEX IS TO BE CONSIDERED
      GOTO 20240
28070 CALL UNIQUE(LONGEX,COLS,FILLED,KRULES)
C     J HAS NOT BEEN TEMPERED WITH. WE WANT TO RETURN THERE
28077 DO 28078 LMN=1,LMIN1
28078 EXENT(LMN,RULE+1)=BUFFER(LMN)
      IRULES=RULE
      LIST(LF,RULE)=MBRNCH
C     THIS WILL GIVE HEADER OF SUB TABLE OF GOTO
28079 IF(ENTRY(J,1).EQ.GOTO.AND.J.NE.NCRD)  GOTO 28550
      LMIN1=L-1
      RULE=RULE+1
C     MUST START THE NEW RULE NOW
      IF(LF.GT.LFF)  LFF=LF
      LF=1
      GOTO 2010
C     IF S.SW 04 ON, INTERMEDIATE OUTPUT WILL BE THERE
C     ULTIMATELY WE HAVE TO KEEP ..GOTO 2010.. INSTEAD OF 20914
C     20914 WILL PROVIDE INTERMEDIATE OUTPUT
```

```
28500 CONTINUE
C       ULTIMATELY KEEP IT AS   IF(JJ.GT.J)  GOTO 28511 OG GOTO 43
C      /109
C       TO BE UTILIZED AT 250+.....
        IF(JJ.GT.J)  GOTO 43109
        IPATH=100
        GOTO 250
C       ULTIMATELY KEEP 43109 AT 28501
28550 RETFRM=PREVNO(J)
        DO 28560 J3=1,NCRD
28560 IF(BCDTAB(J3).EQ.RETFRM)  GOTO 28570
        PAUSE  66666
         ISTMN=28560
        PRINT 24002,ISTMN
        STOP
28570 J=J3
        GOTO 28579
        END

$IBFTC UNIQUE
        SUBROUTINE UNIQUE(LONGEX,COLS,FILLED,IRULES)
        INTEGER LONGEX(20),COLS(80),FILLED,FIRST
C---------------- - -------- ------------------   - ------- ---
C
C       THIS SUB COMPARES ELEMENTS OF LONGEX WITH THOSE
C       OF COLS.  ULTIMATELY ONLY ONCE ANY ELEMENT OF LONGEX
C       APPEARS IN COLS.
C
C------ --------  - ------ -  -------------  -     ------  ---
        IF(FILLED.EQ.0)  GOTO 300
        FIRST=1
50      DO 200 I=FIRST,IRULES
        DO 100 K=1,FILLED
        IF(LONGEX(I).EQ.COLS(K))  GOTO 200
100     CONTINUE
        FILLED=FILLED+1
        COLS(FILLED)=LONGEX(I)
200     CONTINUE
        PRINT 250,(COLS(I),I=1,FILLED)
250     FORMAT(1H ,*CONTENTS OF COLS ARE*/1H ,21A6/)
        RETURN
300     COLS(1)=LONGEX(1)
        FILLED=1
        FIRST=2
        GOTO 50
        END
```

```
$IBFTC MERGE    NOPRNT
       SUBROUTINE MERGE(NOMERG,MBRNCH,LLL,NXTBRN,FOUND,L,TRULES,L
      1ONGEX,KRULES,JCURNT)
C
C            FEB.,1969
C
       INTEGER TRULES ,LONGEX(20)
       INTEGER PLUS
       INTEGER BEGIN,GOTO,STOP,STARS
       INTEGER RETURN
       INTEGER BLANK
       INTEGER FOUND,ELSE ,NUMB(32)
       INTEGER BRNCH(150,3),CONDSN(40,7),ENTRY(150,7),EXENT(40,40
      /),EXCUTD
      1(150,3),N(400),PREVNO(150),STMNT(150),LIST(40,40),RULE,RET
      2,HEADER(100),SUBTAB,KEPTRC(100),CONENT(10),ACTENT(10),ENTR
      /YS(7)
       DATA BLANK,PLUS/1H ,2H++/
       DATA RETURN/6HRETURN/
       COMMON// BRNCH,CONDSN,ENTRY,EXENT,EXCUTD,NCRD,N,PREVNO,STM
      /NT,LIST,
      2LMIN1,LF,LFF,IRULES,RULE,RETFRM
      3 ,HEADER,SUBTAB,NOSTRO,ENTRYS
       COMMON /RULES/ NUMB,ELSE, LOGDT,NOSUBT
       COMMON/DETAB/BEGIN,GOTO,STOP,STARS
C      ----------    --------  ---------------- --+  - ------- ---
C
C      THIS SUBROUTINE IS CALLED WHEN AN ENTRY CORROSPONDING TO A
C      SUBTABLE IS FOUND IN BRNCH AND A CORROSPONDING ..SUB DT..
C       IN ENTRY(L,1).
C      NOMERG IS THE NO. OF TIMES MERGE IS CALLED.
C      KEPTRK IS KEEP TRACK AND AVOIDS DUPLICATION OF
C      SUBTABLES
C      2) SAVES IN TIME
C      LLL IS THE TOTAL (CURRENT) NO. OF ROWS ALREADY ASSEMBLED
C      IN DT
C      LL IS THE CURRENT ROW OF DT UNDER QUESTION
C      L IS THE NO. OF ROWS AT RETURN FROM $MERGE$
C      TRULES IS SIMILAR TO L ABOVE
C      IRULES IS THE TOTAL NO OF RULES COVERED SO FAR.
C
C      ---------    -------- -------------- ---  -- ------- ---
C
       DO 17 I=1,10
 17    CONENT(I)=BLANK
       IF(NOMERG.EQ.1)  GOTO 216
 18    LMIN1=1
       DO 100 I=1,NOMERG
       IF(MBRNCH.EQ.KEPTRC(I))  GOTO 250
```

```
100     CONTINUE
C       THIS BRANCH HAS NOT BEEN $$SENT FOR$$ BEFORE
        DO     200 I=1,NOSUBT
        IF(MBRNCH.EQ.HEADER(I))  GOTO 300
200     CONTINUE
        PRINT 210,MBRNCH
210     FORMAT(//1H ,*MACHINE ERROR IN MERGE,MBRNCH=*,2X,A6//)
        STOP
216     DO 2191 I=1,100
2191    KEPTRC(I)=BLANK
        INMRG=1
        GOTO 18
250     FOUND=1
        TRULES=RULE
        IRULES=RULE
        GOTO 1500
300     IF(I.GT.1)  IMIN1=I-1
        REWIND 4
        ISAVE=I
        IF(I.EQ.1)  GOTO 600
        DO 500 K=1,IMIN1
        READ(4,62985) HEADER(K),NOSTRO,KRULES
62985   FORMAT(A5,2I4)
        DO 400 L=1,NOSTRO
        READ(4,62987) (ENTRYS(J),J=1,5),(CONENT(J),J=1,KRULES)
400     CONTINUE
        READ(4,62989) (ACTENT(J),J=1,KRULES )
500     CONTINUE
62989   FORMAT(*GO TO*,26X,16A6)
62987   FORMAT(5A6,1X,16A6)
C
C       TAPE IS NOW LOCATED PROPERLY
C       READY TO READ THE SUB TABLE UNDER QUESTION
C
600     READ(4,62985) HEADER(I),N6STRO,KRULES
C       -1 BECAUSE OF $$ELSE$$ RULE OF SUBTABLES
        IF( NOSTRO.NE.1)  IRULES=RULE+(KRULES-1)-1
C
C
C       IN CASE OF COMPUTED AND ASSIGNED GOTOS, WE DO NOT HAVE AN
C      /ELSE
C       RULE.
        IF(NOSTRO.EQ.1)  GOTO 7001
        DO 650 I=1,LLL
        DO 650 K=RULE,IRULES
        EXENT(I,K)=LONGEX(I)
C       THIS IS TO CARRY OVER  FROM FOR THE BEGINING OF SUBTABLE
650     CONTINUE
C       TO GET NEW RULE ,IRULES WAS INCREMENTED OIN FORTAB
```

```
      DO 1000 K=1,NOSTRO
7001  READ (4,62987) (ENTRYS(J),J=1,5),(CONENT(J),J=1,KRULES)
      IF(NOSTRO.EQ.1)  GOTO 1001
      DO 800 J=1,LLL
      J1=J
      DO 760 I=1,5
C      INSERTING FOR TESTING
      IF(ENTRYS(I).NE.CONDSN(J,I))  GOTO 800
760   CONTINUE
C
C     A MATCHING CONDITION WHICH WAS ALREADY THERE HAS BEEN FOUN
C    /D
      LL=J
      IPATH=10
      GOTO 900
800   CONTINUE
850   IPATH=1
      LL=LLL+1
      DO 855 I=1,5
855   CONDSN(LL,I)=ENTRYS(I)
C
C     THIS IS THE CASE OF A NEW NEW CONDITION
C
      LLL=LLL+1
900   CONTINUE
      MRULE=0
      DO 870 I=RULE,IRULES
      MRULE=MRULE+1
870   EXENT(LL,I)=CONENT(MRULE)
1000  CONTINUE
1001  READ(4,62989)  (ACTENT(J),J=1,KRULES)
      IF(NOSTRO.EQ.1)  GOTO 1200
      NXTURN=ACTENT(1)
      GOTO 12350
C     INSERTED ON MAY 2,1969 TO SEE ALTERNATIVE
C
2     KMIN1=KRULES-1
      IF(ACTENT(1).EQ.STARS)    GOTO 12356
C     MUST SET IT RIGHT FOR COLS AND LONGEX
      DO 1234 I=1,NCRD
      IF(STMNT(I).EQ.ACTENT(1))  GOTO 12347
1234  CONTINUE
C     THE ACTION STATEMENT BELONGS TO TABLE ..B.. AND IS THUS AN
C    / ACTUAL
C      ACTION
      DO 12346 I=RULE,IRULES
12346 LIST(LF,I)=ACTENT(1)
      LONGEX(1)=ACTENT(1)
      LF=LF+1
```

```
      GOTO 12350
12347 IF(ENTRY(I,1).EQ.STOP)  GOTO 12354
      IF(ENTRY(I,1).EQ.RETURN)  GOTO 12355
C     THEN IT MUST BE LEADING TO ANOTHER IF AND PROCESS MUST BE
C     CONTINUED. SO ENTER IT IN LONGEX
C     IT COULD BE A ..GOTO.. ALS6. DOES NOT MATTER
      IF(ACTENT(1).NE.STARS.AND.STMNT(I).NE.BLANK)  LONGEX(1)=AC
     /TENT(1)
C     OTHERWISE IT WILL BE TAKEN CARE BY FORTAB
      IF(ACTENT(1).NE.STARS.AND.STMNT(I).NE.BLANK)  IGOAL=ACTENT
     /(1)
12348 DO 12349 IJ=RULE,IRULES
12349 LIST(LF,IJ)=IGOAL
12350 FOUND=2
      GOTO 1300
12354 IGOAL=STOP
      LONGEX(1)=0
      GOTO 12348
12355 IGOAL=RETURN
      LONGEX(1)=0
      GOTO 12348
12356 I=JCURNT+1
      LONGEX(1)=0
      WRITE(99,23) PLUS,N(I)
23    FORMAT(A2,A3)
      READ(99,5) IGOAL
5     FORMAT(A5)
      DO 235 K=RULE,IRULES
235   LIST(LF,K)=IGOAL
      GOTO 12347
C     1200 IS THE CASE OF ASSIGNED AND COMPUTED GOTO
C     OTHER WISE TAKE UP $ELSE        ,GOTO PREVNO
C1200  CONSIDER EACH ACTION IN TURN  SET JJ
C     FOR SUBTABLES OF EITHER KIND ,THE NEXT W IS SELECTED
C     AFTER RETURNING TO MAIN PROGRAM
1200  FOUND=3
      DO 1225 I=1,KRULES
1225  LONGEX(I)=ACTENT(I)
C     THIS WILL BE USED AFTER RETURNING TO FORTAB
      KEPTRC(INMRG)=HEADER(ISAVE)
      INMRG=INMRG+1
      TRULES=RULE
C     INCREMENTING WILL BE DONE IN FORTAB
      GOTO 1500
1300  KEPTRC(INMRG)=HEADER(ISAVE)
      INMRG=INMRG+1
1350  TRULES=IRULES
C     TO A/C FOR DECREMENTING. MAKE READY FOR NEW STORAGE
      RULE=IRULES
1500  L=LLL
```

```
      LMIM1=L-1
      RETURN
      END



$IBFTC CMPL
C     OF ..NOOF1S.. AND RETURNS THE ROW NUMBER AS ..KTHROW..
      SUBROUTINE CMPL(WHICH,EXENT,FIRSTC,LASTC,RESULT)
      INTEGER EXENT(40,40),RESULT(40),WHICH
      INTEGER FIRSTC
C*************************************************************************
C
C     SUBRIUTINE CMPL RETURNS THE COMPLIMENT OF ..WHICH..
C     IN RESULT
C*************************************************************************
      DO 100 I=FIRSTC,LASTC
      IF(EXENT(WHICH,I).EQ.1)  GOTO 150
      IF(EXENT(WHICH,I).EQ.0)  GOTO 160
      GOTO 100
150   RESULT(I)=0
      GO TO 100
160   RESULT(I)=1
      GO TO 100
100   CONTINUE
      RETURN
      END
$IBFTC LOWEST
      SUBROUTINE LOWEST(NOOF1S,KTHROW,FIRSTR,LASTR)
      INTEGER NOOF1S(32)
      INTEGER FIRSTR
C*************************************************************************
C
C     LOWEST SUBROUTINE PICKS UP THE ..LOWEST.. NUMBER OUT OF
C     ELEMENTS
C
C*************************************************************************
      KTHROW=FIRSTR
      LOWST=NOOF1S(FIRSTR)
      NEXTR=FIRSTR+1
      DO 100 I=NEXTR,LASTR
      IF(NOOF1S(I).GE.LOWST)  GO TO 100
      LOWST=NOOF1S(I)
      KTHROW=I
100   CONTINUE
      RETURN
      END
```

```
$IBFTC REPLAC
      SUBROUTINE  REPLAC(WHAT,WHERE,WHICH,FIRSTC,LASTC,PUT,FIRST
     /R,LASTR,
     1NOROWS,NOCOLS)
      INTEGER FIRSTC,FIRSTR
      INTEGER WHERE,WHICH,HOWMNY,WHAT(NOROWS,NOCOLS),BUFFER(32),
     /PUT
      INTEGER BUFF2(32),DUMMY(540)
C     DUMMY IS INSERTED TO ECONOMISE IN SPACE. BUFFER OF SUB IS
C     DIFFERENT FROM BUFFER OF ..SOURCE.. IN MAIN
      COMMON /SOURCE/ BUFFER,BUFF2,DUMMY
C*****************************************************************
C
C     SUBROUTINE REPLAC INTERCHANGES TWO ROWS.
C     IN ARRAY WHAT(LLL,HOWMNY) ..WHICH..  ROW IS REPLACED IN ..
C     WHERE
C     ROW ..WHERE.. IS DUMPED IN ROW ..PUT..
C     EXAMPLE-
C        IN EXENT(WHAT), THE KTHROW(WHICH) IS TO REPLACE THE
C     I(WHERE) TH ROW .EACH ROW HAVING KRULES(HOWMNY) ELEMENTS.
C     THE I TH ROW IS TO BE PUT IN K+1 ST ROW(..PUT..).
C     SIZE OF EXENT IS NOROWS,NOCOLS
C
C*****************************************************************
      DO 100 IJ=FIRSTC,LASTC
100   BUFFER(IJ)=WHAT(WHERE,IJ)
      DO 200 IJ=FIRSTC,LASTC
200   WHAT(WHERE,IJ)=WHAT(WHICH,IJ)
      DO 400 IJ=FIRSTC,LASTC
400   BUFF2(IJ)=WHAT(PUT,IJ)
      DO 500 IJ=FIRSTC,LASTC
500   WHAT(WHICH,IJ)=BUFF2(IJ)
      DO 300 IJ=FIRSTC,LASTC
300   WHAT(PUT,IJ)=BUFFER(IJ)
      RETURN
      END
$IBFTC REPCOL
      SUBROUTINE REPCOL(WHAT,WHERE,WHICH,FIRSTR,LASTR,PUT,FIRSTC
     /,LASTC,N
     1 OROWS,NOCOLS)
      INTEGER FIRSTR,FIRSTC
      INTEGER WHERE,WHICH,HOWMNY,WHAT(NOROWS,NOCOLS),BUFFER(32),
     /PUT,BUFF
     12(32),COLS,DUMMY(540)
C     DUMMY IS INSERTED TO ECONOMISE IN SPACE. BUFFER OF SUB IS
C     DIFFERENT FROM BUFFER OF ..SOURCE.. IN MAIN
      COMMON /SOURCE/ BUFFER,BUFF2,DUMMY
C*****************************************************************
C
C     SUBROUTINE REPCOL INTERCHANGES TWO COLUMNS.
```

```
C       IN ARRAY(NOROWS,NOCOLS)  ..WHICH.. COL. IS REPLACED IN ..
C       ..WHERE..
C       COL. ..WHERE.. IS DUMPED IN .. PUT ..
C       .......FOR CLARIFICATIONS  PL. SEE SUB. REPLAC
C
C*******************************************************************
        DO 100 IJ=FIRSTR,LASTR
100     BUFFER(IJ)=WHAT(IJ,WHERE)
        DO 200 IJ=FIRSTR,LASTR
200     WHAT(IJ,WHERE)=WHAT(IJ,WHICH)
        DO 400 IJ=FIRSTR,LASTR
400     BUFF2(IJ)=WHAT(IJ,PUT)
        DO 500 IJ=FIRSTR,LASTR
500     WHAT(IJ,WHICH)=BUFF2(IJ)
        DO 300 IJ=FIRSTR,LASTR
300     WHAT(IJ,PUT)=BUFFER(IJ)
        RETURN
        END
$IBFTC PARSEM
        SUBROUTINE  PARSEM
        INTEGER BRNCH(150,3),CONDSN(40,7),ENTRY(150,7),EXENT(40,40
      /),EXCUTD
      1(150,3),N(400),PREVNO(150),STMNT(150),LIST(40,40),RULE,RET
      /FRM
      2,HEADER(100),SUBTAB,ENTRYS(7)
        INTEGER FIRSTR,FIRSTC,TDTRB,TDTRE,TDTCB,TDTCE,BDTRB,BDTRE,
      1BDTCB,BDTCE,ACTSTB(40,5)
        COMMON/SPARSE/NOACT,ACTSTB
C       NOACT IS NO OF ACTION STUBS.ACTSTB IS FOR ACTION STUB
        COMMON//BRNCH,CONDSN,ENTRY,EXENT,EXCUTD,NCRD,N,PREVNO,STMN
      /T,LIST,
      2LMIN1,LF,LFF,IRULES,RULE,RETFRM
      3 ,HEADER,SUBTAB,NOSTRO,ENTRYS
        READ 10,LLL,IRULES
 10     FORMAT(3I2)
        DO 30 J=1,LLL
        READ 20,(CONDSN(J,I),I=1,5),(EXENT(J,IJ),IJ=1,IRULES
      / )
        PRINT21,(CONDSN(J,I),I=1,5),(EXENT(J,IJ),IJ=1,IRULES
      / )
 20     FORMAT(5A3,16A4)
 21     FORMAT(1H ,5A3,16A4)
 30     CONTINUE
        DO 35 J=1,NOACT
        READ 20,(ACTSTB(J,I),I=1,5),(LIST(J,I),I=1,IRULES)
 35     PRINT21,(ACTSTB(J,I),I=1,5),(LIST(J,I),I=1,IRULES)
        PRINT 60
 60     FORMAT(////)
        CALL PARSE(1,LLL,1,IRULES,TDTRB,TDTRE,TDTCB,TDTCE,BDTRB,BD
      1TRE,BDTCB,BDTCE)
        DO 40 J=1,LLL
```

```
40      PRINT21,(CONDSN(J,I),I=1,5),(EXENT(J,IJ),IJ=1,IRULES
       / )
        DO 41 J=1,NOACT
41      PRINT 21,(ACTSTB(J,I),I=1,5),(LIST(J,I),I=1,IRULES)
        PRINT 60
        CALL PARSE(BDTRB,BDTRE,BDTCB,BDTCE,TDTRB,TDTRE,TDTCB,TDTCE
       1,BDTRB,BDTRE,BDTCB,BDTCE)
        DO 50 J=1,LLL
50      PRINT21,(CONDSN(J,I),I=1,5),(EXENT(J,IJ),IJ=1,IRULES
       / )
        PRINT 60
         STOP
        END
$IBFTC PARSE
        SUBROUTINE PARSE(FIRSTR,LASTR,FIRSTC,LASTC,TDTRB,TDTRE,TDT
       /CB,TDTCE
       1,BDTRB,BDTRE,BDTCB,BDTCE)
        INTEGER ACTSTB(40,5)
        INTEGER FIRSTR,FIRSTC,TDTRB,TDTRE,TDTCB,TDTCE,BDTRB,BDTRE,
       /BDTCB,BD
       1TCE
C--------------  - --------  ------------------  -- -------  ---
C       FIRSTR= BEGINING OF DT ROW WISE
C       LASTR=END OF        DT COL WISE
C       SIMILARLY FOR FIRSTC AND LASTC  ..C.. STANDING FOR COL.
C       OTHER ARGUMENTS MEAN 1ST CHAR. IS T OR B..T FOR TOP AND B
C       FOR BOTT.
C       OM SUB D.T.
C       LAST CHAR IS B OR E . B FOR BEGIN AND E FOR END
C       LAST BUT ONE CHAR IS C OR 9 . C FOR COLUMN AND R FOR ROW
C--------------        --------  ---------------  --  --  -----------
        INTEGER BLANK,NOOF1S(32),SIM LAR(32),KBAR(32),DUMMY(412),R
       /EST(32)
       1,TRACK(32,1),BUFF2(32)
C       BECAUSE BUFFER IS BEING USED TO A/C FORVA NO OF VARIABLES,
C      /HENCE
C       TO MAKE FOR THE DIFFERENCE, INSERTING DUMMU
        INTEGER BRNCH(150,3),CONDSN(40,7),ENTRY(150,7),EXENT(40,40
       /),EXCUTD
       1(150,3),N(400),PREVNO(150),STMNT(150),LIST(40,40),RULE,RET
       /FRM
       2,HEADER(100),SUBTAB,KEPTRC(100),CONENT(10),ACTENT(10),ENTR
       /YS(7)
        DATA BLANK/4H    /
        COMMON// BRNCH,CONDSN,ENTRY,EXENT,EXCUTD,NCRD,N,PREVNO,STM
       /NT,LIST,
       2LMIN1,LF,LFF,IRULES,RULE,RETFRM
       3 ,HEADER,SUBTAB,NOSTRO,ENTRYS
        COMMON/SPARSE/NOACT,ACTSTB
C       COMMON /SOURCE/ NOOF1S,SIMLAR,KBAR,REST,TRACK,BUFF2,DUMMY
```

```
C      DUMMY IS INSERTED TO ECONO4ISE SPACE.BUFFER OF SUB IS DIFF
C     /ERENT
C      FROM BUFFER OF ..SOURCE.. IN MAIN.
C*****************************************************************
C
C      PARSE SUB. DOES THE PARSING OF A LARGE DT.LLL MUST BE
C      SPECIFIED THROUGH ARGUMENT LIST.NO OF RULES ..IRULES.. IS
C      LINKED THROU. COMMON BLOCK//.
C      LLL IS THE NUMBER OF CONDITIONS IN DT.C
C      RETURNED VALUE OF FOUND IS 100 FOR PARSING POSSIBLE,OTHERW
C     /ISE 0
C
C*****************************************************************
       LLL=LASTR-FIRSTR+1
       REWIND 3
       DO 20 I=1,32
       DO 20 J=1,1
20     TRACK(I,1)=I
       DO 50 I=1,32
50     NOOF1S(I)=0
       DO 100    I=FIRSTR,LASTR
100    WRITE(3,110) (CONDSN(I,IJ),IJ=1,5),(EXENT(I,IJ),IJ=FIRSTC,
      /LASTC)
       DO 101 I=1,NOACT
101    WRITE(3,110) (ACTSTB(I,IJ),IJ=1,5),(LIST(I,IJ),IJ=FIRSTC,L
      /ASTC)
       REWIND  3
C      THIS IS TO SAVE DT FOR RESTORING LATER
110    FORMAT(5A6,1X,16A6)
C      INSERT DO LOOP FOR ACTION SET
       DO 200    I=FIRSTR,LASTR
       DO 200 IJ=FIRSTC,LASTC
       IF(EXENT(I,IJ).EQ.BLANK)  GOTO  150
       IF(EXENT(I,IJ).NE.BLANK)  GOTO 160
       GOTO 200
150    EXENT(I,IJ)=1
       GOTO 200
160    EXENT(I,IJ)=0
       GOTO 200
200    CONTINUE
       DO 300    I=FIRSTR,LASTR
       DO 300 IJ=FIRSTC,LASTC
       IF(EXENT(I,IJ).EQ.1)  NOOF1S(I )=NOOF1S( I)+1
300    CONTINUE
       CALL LOWEST(NOOF1S,KTHROW,FIRSTR,LASTR)
       IF(NOOF1S(KTHROW).EQ.0.AND.KTHROW.EQ.FIRSTR)  GOTO 850
       IF(NOOF1S(KTHROW).EQ.0)  GOTO 801
       IPATH=2
C      TO KEEP TRACK OF DIFFERENT PARTS OF TREE
```

```
      K=0
      DO 400    I=FIRSTR,LASTR
      IF(I.EQ.KTHROW)   GOTO 400
      DO 375 IJ=FIRSTC,LASTC
      IF(EXENT(KTHROW,IJ).NE.1)   GOTO 375
      IF(EXENT(I,IJ).NE.1)   GOTO 400
375   CONTINUE
      K=K+1
      SIM LAR(K)=I
400   CONTINUE
      NOROWT=K+1
C     +1 TO ACCOUNT FOR KTHROW
      TDTRB=FIRSTR
      TDTRE=FIRSTR+K+1-1
      BDTRB=TDTRE+1
      BDTRE=LASTR
      CALL CMPL(KTHROW,EXENT,FIRSTC,LASTC,KBAR)
      IJKL=1
C     GET THE COMPLIMENT OF KTH ROW OF DT EXTENDED ENTRIES.
C     RETURNED IS KBAR.NOT DT IS AT PRESENT REDUCED TO BOOLEAN
C     MATRIX
C     BOOLEAN MATRIX
      DO 500    I=FIRSTR,LASTR
      DO 50070 IJ=1,K
      IF(I.EQ.SIM LAR(IJ).OR.I.EQ.KTHROW)   GOTO 50070
      IJKL=2
      CALL CMPL(I,EXENT,FIRSTC,LASTC,REST)
70010 FORMAT(/1H ,5A6,16A6)
      DO 50050 IJK=FIRSTC,LASTC
      IF(KBAR(IJK)*REST(IJK).EQ.0)   GOTO 50050
      GO TO 500
50050 CONTINUE
50070 CONTINUE
      GOTO 520
500   CONTINUE
      GO TO 900
520   CONTINUE
      PRINT 50071
50071 FORMAT(//1H ,*PARSING POSSIBLE*//)
      IJKL=3
      I=1
      CALL REPLAC(EXENT,I,KTHROW,FIRSTC,LASTC,K+1,FIRSTR,LASTR,4
     /0,40)
C---------------    -------- --------------- ---   -- ----------
C     ARGUMENTS MEAN RESPECTIVELY-
C     EXENT       WHAT TO REPLACE I.E. ARRAY NAME FROM WHICH TO
C     PICKUP
C     I           WHERE TO REPLACE
C     KTHROW      WHICH TO REPLACE
C     FIRSTC AND LASTC GIVE THE ELEMENTS OF KTHROW TO BE
C     CONSIDERED
```

```
C       K+1            WHERE TO DUMP I TH ROW
C       LLL            TOTAL SIZE OF EXENT I.E EXENT(40,40), TO
C       HANDLED BY REPLACE
C       40,40 IS THE ACTUAL NO. OF ROWS AND ACTUAL NO. OF COLS.
C
C---------------   --------  -----------+--  --   - ------- ---
        CALL REPLAC(CONDSN,I,KTHROW,1,5,K+1,FIRSTR,LASTR,32,7)
        CALL REPLAC(TRACK,I,KTHROW,1,1,K+1,FIRSTR,LASTR,32,1)
C
C       THIS IS TO KEEP TRACK OF REARRANGED ROWS
C       IF KTHROW=1  ABOVE TWO CALLS NEED NOT BE INSERTED
        KRULES=IRULES
        IJKL=1
        DO 600 J=1,K
        J1=J+1
        KJ1=K+J+1
        I=SIMLAR(J)
        CALL REPLAC(CONDSN,J+1,I,1,5,K+J+1,FIRSTR,LASTR,32,7)
        CALL REPLAC(TRACK,J+1,I,1,1,K+J+1,FIRSTR,LASTR,32,1)
C       THIS IS TO KEEP TRACK OF REARRANGED ROWS
        CALL REPLAC(EXENT,J+1,I,FIRSTC,LASTC,K+J+1,FIRSTR,LASTR,40
       /,40)
600     CONTINUE
        KPLUS1=K+1
        GO TO 1000
10020   KPLUS1=K+1
C       NEXT LOOP MAY NOT BE NEEDED AT ALL
C
        KPLUS2=K+2
        LLLMK1=LLL-(K+1)
        I=KPLUS2
        KJ1=KJ1+1
        DO 800    J=FIRSTR,LASTR
        DO 800 M=1,K
        IF(J.EQ.KTHROW.OR.J.EQ.SIMLAR(M))  GOTO 800
        CALL REPLAC(CONDSN,KJ1,I,1,5,K+I+1,FIRSTR,LASTR,32,7)
        CALL REPLAC(EXENT,KJ1,I,FIRSTC,LASTC,K+I+1,FIRSTR,LASTR,40
       /,40)
        CALL REPLAC(TRACK,KJ1,I,1,1,K+I+1,FIRSTR,LASTR,32,1)
        KJ1=KJ1+1
        I=I+1
800     CONTINUE
1000    CONTINUE
C       AT THIS STAGE THE CHANGED BOOLEAN DT IS AVAILABLE
C       THE ORIGINAL IS TO OBTAINED FROM NOW BY READJUSTMENT
        KRULES=LASTC-FIRSTC+1+5
        DO 2000   I=FIRSTR,LASTR
        READ(3,110) (DUMMY(IJ),IJ=1,5),(DUMMY(IJ),IJ=6,KRULES)
        DO 2500IJK=FIRSTR,LASTR
        ITJ=TRACK(IJK,1)
```

```
      IF(I.EQ.TRACK(IJK,1))  GOTO 2600
2500  CONTINUE
      ISTMN=2500
      PRINT 2510,ISTMN
2510  FORMAT(//1H ,*MACHINE ERROR *,I5)
      STOP
2600  DO 2650 IJ=FIRSTC,LASTC
      IF(EXENT (IJK,IJ).EQ.1) GOTO 2670
      IF(EXENT (IJK,IJ).EQ.0)  GOTO 2680
      ISTMN=2600
      PRINT 2510,ISTMN
      STOP
2670  EXENT (IJK,IJ)=BLANK
      GOTO 2650
2680  EXENT (IJK,IJ)=DUMMY(IJ+5)
2650  CONTINUE
2000  CONTINUE
      KPLUS=FIRSTR+(K+1)-1
      NOROWB=LLL-NOROWT
      IF(IPATH.EQ.1)  GOTO 3350
C     SKIP THE NEXT FEW STEPS BECAUSE SIZE OF TOP DT IS ALREADY
C    /KNOWN
C     FOR PATH 1
      M=1
      NN=0
C     GET SIZE OF TOP LEFT SUB DT.
C     NO. OF ROWS IS ALREADY KNOWN TO BE KPLUS1=NOROWT
      DO 3300 I=FIRSTC,LASTC
      DO 3200 J=FIRSTR,KPLUS
      IF(EXENT(J,I).NE.BLANK)  G6TO 3300
3200  CONTINUE
      NN=NN+1
      SIMLAR(M)=I
      M=M+1
3300  CONTINUE
3350  CONTINUE
      NOCOLT=NN
C     NOCOLB=IRULES-NOCOLT
      NOCOLB=LASTC-FIRSTC+1-NOCOLT
      IF(IPATH.EQ.1)  GOTO 3501
C     NO NEED OF SHIFTING THE COLS. FOR PATH 1
      NPLUS1=NN+1
      IL=1
      IR=NPLUS1
C     IL IS THE CURRENT ROW ON THE LEFT WHICH VAN BE FILLED UP.
C    /IR IS ON
C     THE RIGHT
      DO 3500 IJ=FIRSTC,LASTC
      DO 3400 IJK=1,NN
```

```
      IF(IJ.EQ.SIMLAR(IJK))  GOTO 3500
3400  CONTINUE
      GOTO 3450
3470  IL=IL+1
      GOTO 3500
3450  IF(IJ.EQ.1)  GOTO 3470
C
C     DONOT DISTURB THE FIRST COLUMN IF IT BELONGS TO TOP LEFT S
C    /UB DT
      CALL REPCOL(EXENT,IL,IJ,FIRSTR,LASTR,IR,FIRSTC,LASTC,40,40
     /)
C
C     THIS IS TO REPLACE THE COLS. NOTE REPLAC IS FOR ..ROWS.. A
C    /ND
C     REPCOL IS FOR ..COLUMNS..
      CALL REPCOL(LIST,IL,IJ,1,LFF,IR,FIRSTC,LASTC,40,32)
C     INSERT XXX
C     INSERT FOR ACTION ENTRIES
C     LIST CAN BE USED INSTEAD OF ACTENT
      IR=IR+1
      GO TO 3470
3500  CONTINUE
      TDTCB=FIRSTC
      TDTCE=FIRSTC+NN-1
      BDTCB=TDTCE+1
      BDTCE=LASTC
3501  CONTINUE
C     INSERT FOR GETTING THE CHANGED DT
      FOUND=100
      RETURN
801   CALL REPLAC(EXENT ,FIRSTR,KTHROW,FIRSTC,LASTC,KTHROW,FIRST
     1R,LASTR,40,40)
      CALL REPLAC(TRACK,FIRSTR,KTHROW,1,1,KTHROW,FIRSTR,LASTR,32
     /,1)
      CALL REPLAC(CONDSN,FIRSTR,KTHROW,FIRSTC,LASTC,KTHROW,FIRST
     1R,LASTR,32,7)
850   TDTRB=FIRSTR
      TDTRE=FIRSTR
      TDTCB=FIRSTC
      TDTCE=LASTC
      BDTRB=FIRSTR+1
      BDTRE=LASTR
      BDTCB=FIRSTC
      BDTCE=LASTC
      NN=LASTC-FIRSTC+1
      KPLUS1=1
      IPATH=1
C     TO KEEP TRACK OF DIFFERENT PARTS OF THE TREE
      NOROWT=1
      K=1
```

```
      GOTO 1000
900   PRINT 910
910   FORMAT(//1H ,*PARSING NOT 70SSIBLE*// )
      FOUND=0
      RETURN
      END
$IBFTC GETENT  NOPRNT
      SUBROUTINE   GETENT(ENTRYS,MAXNOP)
      INTEGER  BUFFER(600),ENTRYS(20),FINAL,FARMAT(5),FULWR4,RIG
     /HTP,
     1REMAIN,MAXNOP,UPLIMT,UPTO ,UNPAIR
      INTEGER FOUND
      DATA LEFTP,RIGHTP,FARMAT(1),FARMAT(3),FARMAT(5)/1H(,1H),1H
     /(,4HA6,A
     1,1H)/
      COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
C ******************************************************************
C     THIS SUBROUTINE IS TO BE USED FOR IFS AND COMPUTED GOTOS
C     TO GET THE CHARACTER SET WITHIN THE MATCHING PARANTHISIS
C     THAT IS WELL FORMED EXPRESSIONS.IT ASSUMES THAT LAST
C     CHARACTER ENCOUNTERED IS A ( AND THE NEXT CH. OF BUFFER
C     IS AT INITAL. WHEN RETURNING LAST CHARACTER IS AT FINAL
C     ) OF EXPRESSION)
C *   FOUND WILL BE USEFUL FOR CONTINUATION CARDS OF IF
C ******************************************************************
      MAXNOP=1
      UNPAIR=1
      UPTO=INITAL+UPLIMT-1
      DO  100 I=INITAL,UPTO
      IF(BUFFER(I).EQ.LEFTP)  GO TO 150
      IF(BUFFER(I).EQ.RIGHTP)UNPAIR=UNPAIR-1
200   IF(UNPAIR.EQ.0)  GO TO 500
      CO TO 100
150   UNPAIR=UNPAIR+1
      IF(MAXNOP.LT.UNPAIR)  MAXN6P=UNPAIR
      GO TO 200
100   CONTINUE
      FOUND=0
      RETURN
500   K=I-INITAL
C     HERE WE DONOT HAVE TO ADD 1 TO GET PROPER K
      FINAL=I-1
      WRITE(99,600) (BUFFER(I),I=INITAL,FINAL)
600   FORMAT(80A1)
      FULWRD=K/6
      REMAIN=K-FULWRD*6
      CALL  BINBCD(FULWRD,FARMAT(2))
      CALL  BINBCD(REMAIN,FARMAT(4))
CFARMAT  FORMAT(14A6,A4)         EXAMPLE
      IF(REMAIN.GT.0)  FULWRD=FULWRD+1
```

```
      READ(99,FARMAT) (ENTRYS(KK),KK=1,FULWRD)
      FOUND=100
      RETURN
      END
$IBFTC RIGHTJ
      SUBROUTINE RIGHTJ(SENT,SENTB,CHARR,CHARI)
C     SENT SHOULD BE LEFT JUSTIFIED  BEFORE BEING SENT TO RIGHTJ
      INTEGER SENT,SENTB,CHARR,CHARI,BUFFER(6),BUFF2(6)
      WRITE(99,6) SENT
6     FORMAT(A6)
      READ(99,61) BUFF2
61    FORMAT(6A1)
      K=0
      DO 200 I=1,6
      IF(BUFF2(I).EQ.CHARR)  GOTO 250
      GOTO 200
250   K=K+1
C     K GIVES THE NO OF CHARACTERS TO BE REMOVED
200   CONTINUE
      NOCHRT=6-K
C      NOCHRT IS NO OF CHAR. TO BE RETAINED
      DO 100 I=1,K
100   BUFFER(I)=CHARI
      DO 300 I=1,NOCHRT
      KI=K+I
300   BUFFER(KI)=BUFF2(I)
      WRITE(99,61) BUFFER
      READ(99,6) SENTB
      RETURN
      END
$IBFTC SEQUNC  NOPRNT,DECK
      SUBROUTINE SEQUNC(LIST,M,JPREV)
      INTEGER BUFFER(600),LIST(20),DLIMTR(10)
      INTEGER FOUND,FINAL,UPLIMT
      DATA DLIMTR /1H.,1H(,1H),1H,,1H=,1H*,1H-,1H+,1H/,1H /
      COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
C     FROM EFFICENCY POINT OF VIEW DELIMETERS SHOULD BE ARRANGED
C     IN THE SEQUENCE OF THEIR ORDER OF OCCURANCE
C     FOR A GIVEN BUFFER IT WILL GIVE US
C     FOR A GIVEN BUFFER IT WILL GIVE US  THE LIST OF DELIMETERS
C     THAT APPEAR BETWEEN COLS. 7-72 FROM LEFT TO RIGHT.ON ADHOC
C     BASIS THE TOTAL NO. OF DELIMITERS IS TAKEN AS 30 WHICH
C     WILL FORM LIST
      DO 20 I=1,20
20    LIST(I)=0
      KK=INITAL
      M=1
50    DO  100 K=KK,JPREV
      DO 100 J=1,9
      IF(BUFFER(K).EQ.DLIMTR(J))  GO TO 200
```

```
      IF(BUFFER(K).EQ.DLIMTR(10))  GOTO 400
      IF(M.GT.20)  GO TO 300
100   CONTINUE
      IF(M.GE.2)  GOTO 400
      RETURN
C
400   M=M-1
      RETURN
200   LIST(M)=DLIMTR(J)
      M=M+1
      KK=K+1
      GO TO 50
300   PRINT 350
350   FORMAT(/1H ,*NO OF DELIMITERS HAS EXCEEDED THE STIPULATED
     1NUMBER OF 20  */)
      RETURN
      END
$IBFTC BINBCD  NODECK,NOPRNT
      SUBROUTINE BINBCD(BINNO,EQBCD)
C     SUB FOR CONVERTING BINNO TO BCDNO*********
      INTEGER  BINNO,EQBCD
      DATA I8X10/010000000000 /
      DATA I8X2,I8X4,I8X6,I8X8/0100,010000,01000000,0100000000/
      EQBCD=MOD(BINNO,100000)/10000*I8X8+MOD(BINNO,10000)/1000*I
     /8X6+
     1 MOD(BINNO,1000)/100*I8X4+4OD(BINNO,100)/10*I8X2+MOD(BINNO
     /,10)
     2 +MOD(BINNO,1000000)/100000*I8X10
      RETURN
      END
$IBFTC REMLBL  NOPRNT
      SUBROUTINE  REMLBL(STMNT,CHAR)
      INTEGER  STMNT,COLS(6),BLA5K,BUFFER(6),CHAR
      DATA BLANK/1H /
C     THIS SUBROUTINE REMOVES THE SUPERFLUOUS CHARACTERS ON THE
C     LEFT
C     IT LEFT JUSTIFIES A QUANTITY
      DO  50 I=1,6
50    BUFFER(I)=BLANK
      WRITE(99,100) STMNT
100   FORMAT(A6)
      READ(99,150) COLS
150   FORMAT(6A1)
      DO  200 I=1,6
      IF(COLS(I).NE.CHAR)  GO TO 300
200   CONTINUE
      STOP
300   KFINAL=6-I+1
      DO  400 K=1,KFINAL
```

```
      BUFFER(K)=COLS(I)
      I=I+1
4)0   CONTINUE
      WRITE(99,150) BUFFER
      READ(99,100) STMNT
      RETURN
      END
$IBFTC GETCON  NOPRNT
      SUBROUTINE GETCON(NAME,KK)
      INTEGER BUFFER(600),INITAL,FINAL,UPLIMT,UPTO,NAME(KK),FARM
     1AT(5),FULWRD,REMAIN
      INTEGER BLANKS
      COMMON /SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
      DATA  FARMAT(1),FARMAT(3),FARMAT(5)/1H(,4HA6,A,1H)/
      DATA BLANKS/5H      /
      DO  200 K=1,KK
200   NAME(K)=BLANKS
      UPTO=INITAL+UPLIMT-1
      WRITE(99,300) (BUFFER(I),I=INITAL,UPTO)
300   FORMAT(80A1)
      FULWRD=UPLIMT/6
      REMAIN=UPLIMT-FULWRD*6
      CALL  BINBCD(FULWRD,FARMAT(2))
      CALL  BINBCD(REMAIN,FARMAT(4))
      IF(REMAIN.NE.0)  FULWRD=FULWRD+1
      READ(99,FARMAT) (NAME(K),K=1,FULWRD)
      RETURN
      END
$IBFTC SQUEZE  NOPRNT
      SUBROUTINE  SQUEZE(COLS,JPREV)
      INTEGER COLS(80),BUFFER(600) ,BLANK,DATA,FORMAT,D,F
      INTEGER FOUND
      DATA BLANK,DATA,FORMAT,D,F/1H ,4HDATA,6HFORMAT,1HD,1HF/
      COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
100   FORMAT(80A1)
      IF(FOUND.EQ.0) GO TO 200
C     WHILE ENTERING SQUEZE FOUND IS 100 ONLY IF WE ARE DEALING
C     WITH A CO TINUATION CARD
      J=JPREV+1
      GO TO 250
200   J=7+JPREV
      DO  111  IJK=1,80
1.1   BUFFER(IJK)=BLANK
250   JSTART=J
      DO  10211  I=7,72
      IF(COLS(I).EQ.BLANK)  GO TO 10211
      BUFFER(J)=COLS(I)
      J=J+1
      IF((BUFFER(7).EQ.D.AND.J.GT.10).OR.(BUFFER(7).EQ.F.AND.J.G
```

```
     1T.12))  GOTO 10200
      GO TO 10211
10200 IF(BUFFER(7).EQ.D)  GO TO 10201
      K=12
      GO TO 10202
10201 K=10
10202 WRITE(99,10203)  (BUFFER(KK),KK=7,K)
10203 FORMAT(6A1)
      READ(99,10204)  IDATA
10204 FORMAT(A4)
      IF(IDATA.EQ.DATA)  GO TO 10213
      READ(99,10205)  IFORMT
10205 FORMAT(A6)
      IF(IFORMT.EQ.FORMAT)  GO TO 10213
10211 CONTINUE
      FOUND=100
      JPREV=J-1
10212 FORMAT(1H ,72A1)
      RETURN
10213 FOUND=0
      RETURN
      END
$IBFTC PUTBRN  NOPRNT
      SUBROUTINE  PUTBRN(OPRATR,GOAL,BRNCH1,BRNCH2,BRNCH3,  N)
C *****************************************************************
C
C
C      THIS SUBROUTINE PUTS THE PROPER VALUE IN BRANCHES
C
C
C *****************************************************************
      INTEGER BUFFER(600)
      INTEGER FOUND,FINAL,UPLIMT
      INTEGER ORR,ANDD,NOTT
      INTEGER OPRATR,GOAL,BRNCH1,BRNCH2,BRNCH3,LEZ,LTZ,EQZ,GEZ,G
     1TZ,NEZ,STARS
      DATA  LEZ,LTZ,EQZ,GEZ,NEZ,STARS,GTZ/
     1 4H.LE.,4H.LT.,4H.EQ.,4H.GE.,4H.NE.,5HXXXXX,4H.GT./
      DATA  ORR,ANDD,NOTT/4H.OR.,5H.AND.,5H.NOT./
C     NOTE THAT AND  NOT ETC ARE ALL 4 CHARACTERS LONG
      COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
      IF(OPRATR.EQ.LEZ)  GO TO 512
      IF(OPRATR.EQ.LTZ)  GO TO 510
      IF(OPRATR.EQ.EQZ)  GO TO 520
      IF(OPRATR.EQ.GEZ)  GO TO 523
      IF(OPRATR.EQ.GTZ)  GO TO 530
      IF(OPRATR.EQ.NEZ)  GO TO 513
      IF(OPRATR.EQ.ORR)  GOTO400
      IF(OPRATR.EQ.NOTT)  GO TO 450
      IF(OPRATR.EQ.ANDD)  GO TO 470
```

```
      FOUND=4
500   CONTINUE
      RETURN
400   FOUND=2
C     FOUND IS RESPECTIVELY 1 FOR AND,2 FOR OR,3 FOR NOT AND
C     4 FOR ILLEGAL OPCODE
C     FOUND IS 100 FOR RELATIONAL OPS
      GOTO500
450   FOUND=3
      GOTO500
470   FOUND=1
      GOTO500
510   BRNCH1=GOAL
      BRNCH2=STARS
      BRNCH3=STARS
      GO TO 780
512   BRNCH1=GOAL
      BRNCH2=GOAL
      BRNCH3=STARS
      GO TO 780
513   BRNCH1=GOAL
      BRNCH3=GOAL
      BRNCH2=STARS
      GO TO 780
520   BRNCH2=GOAL
      BRNCH1=STARS
      BRNCH3=STARS
      GO TO 780
523   BRNCH2=GOAL
      BRNCH3=GOAL
      BRNCH1=STARS
      GO TO 780
530   BRNCH3=GOAL
      BRNCH2=STARS
      BRNCH1=STARS
      GO TO 780
780   FOUND=100
      RETURN
      END
$IBFTC SEARCH  NOPRNT
      SUBROUTINE  SEARCH(DLIMIT,PREVCH,KK,OBTAIN)
      INTEGER  BUFFER(600),DLIMIT,FINAL,OBTAIN,PREVCH(KK),FARMAT
     /(3)
      INTEGER PERIOD
      INTEGER FOUND,UPLIMT,UPTO
      COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
      DATA  FARMAT(1),FARMAT(3)/2H(A,1H)/
      DATA PERIOD/1H./
C**********************************************************************
```

```
C       USED FOR GETTING AND SEPARATING FOR EXAMPLE GOAL OF AN IF
C       OR BRANCH OF AN IF. INITAL IS THE VALUE OF BUFFER AT
C       SEARCH STARTS FOR GETTING OBTAIN,FINAL WHEN OBTAINED.
C       PREVCH CAN ,FOR EXAMPLE  BE ONLY NUMBERS IN GOAL OF GOTO
C       IT DOES NOT SEARCH FOR VIRTUAL DELIMITERS
C***********************************************************************
        UPTO=INITAL+UPLIMT-1
        DO  100 I=INITAL,UPTO
        DO  100  JK=1,KK
        IF(BUFFER(I).EQ.PREVCH(JK).AND.BUFFER(I+1).EQ.DLIMIT)  GO
       /TO 200
        IF(BUFFER(I).EQ.PERIOD.AND.BUFFER(I+1).EQ.DLIMIT) GOTO200
100     CONTINUE
        FOUND=0
        OBTAIN=0
        RETURN
200     K=I-INITAL+1
        FOUND=100
        FINAL=I
        WRITE(99,250) (BUFFER(I),I=INITAL,FINAL)
250     FORMAT(6A1)
        CALL  BINBCD(K,FARMAT(2))
        READ  (99,FARMAT)  OBTAIN
CFARMAT  FORMAT(AN)
        RETURN
        END
$IBFTC BCDBIN  NODECK,NOPRNT
        SUBROUTINE BCDBIN(BCDNO,BINNO)
C       SUB. FOR CONVERTING BCD NUMER TO BINARY
C       NOTE THAT BCDNO MUST  NOT CONTAIN BLANKS IN THE BYTE
C       BLANKS ARE TAKEN AS 60
        INTEGER BCDNO,BINNO
        DATA I8X2,I8X4,I8X6,I8X8/0100,010000,01000000,0100000000/
        BINNO=BCDNO/I8X8*10000+MOD(BCDNO,I8X8)/I8X6*1000+MOD(BCDNO
       /,I8X6)/
       1I8X4*100+MOD(BCDNO,I8X4)/I8X2*10 +MOD(BCDNO,I8X2)
        RETURN
        END
$IBFTC NODLMT  NOPRNT
        SUBROUTINE  NODLMT(NXTQAN,KK,PREVCH,KL,OBTAIN)
        INTEGER BUFFER(600),FINAL,FARMAT(3),NXTCAN(KK),PREVCH(KL),
       /OBTAIN
        INTEGER  FOUND,UPLIMT,UPTO
        COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
        DATA  FARMAT(1),FARMAT(3)/2H(A,1H)/
C ***********************************************************************
C       THIS SUBROUTINE SEARCHES FOR A QUQNTITY OBTAIN WHICH
C       IS FOLLOWED BY A CHARACTER CAN FORM A SET OF CHARACTERS
C       AND NOT A  SINGLE  CHARACTER.
C       EXAMPLE- AFTER RANGE IN DO , INDEX CAN ONLY ALPHABETIC
C       CHARACTER. FOUND IF SEARCH IS SUCESSFUL, IF NOT 0.
C ***********************************************************************
```

```
      UPTO=INITAL+UPLIMT-1
      DO   100 I=INITAL,UPTO
      DO   100 JKL=1,KL
      DO   100  JKK=1,KK
      IF(BUFFER(I).EQ.PREVCH(JKL).AND.BUFFER(I+1).EQ.NXTQAN(JKK)
     1)  GOTO   200
100   CONTINUE
      OBTAIN=0
      FOUND=0
      RETURN
200   K=I-INITAL      +1
      FINAL=I
      FOUND=100
      WRITE(99,250) (BUFFER(I),I=INITAL,FINAL)
      CALL  BINBCD(K,FARMAT(2))
      READ  (99,FARMAT)  OBTAIN
C FARMAT  FORMAT(AN)
250   FORMAT(6A1)
      RETURN
      END
$IBFTC BLOCK   NOPRNT
      BLOCK  DATA
      INTEGER ELSE
      INTEGER NUMB(32)
      INTEGER BEGIN,GOTO,STARS,STOP
      COMMON /RULES/ NUMB,ELSE, LOGDT,NOSUBT
      COMMON/DETAB/BEGIN,GOTO,STOP ,STARS
      DATA BEGIN,STOP,GOTO,STARS/5HBEGIN,6HSTOP  ,6HGOTO  ,5HXXX
     /XX/
      COMMON/OPS/LTZ,LEZ,EQZ,GEZ,GTZ,NEZ
C     KEEP A SIMILAR CARD IN PUTBRN  AND BLOCK
      DATA  NUMB/1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
     120,21,22,23,24,25,26,27,28,29,30,31,32/,LOGDT/6HSUB DT/
     2ELSE/4HELSE/
      END
$IBFTC DTENT   NOPRNT
      SUBROUTINE DTENT(EXENT,RULE,RETFRM,NCRD,ENTRY,LMIN1,LF,CON
     /DSN,
     1PREVNO,LFF,LIST,STMNT,HEADER)
C     PREVNO IS NOW A LOCAL VARIABLE
      INTEGER SAVERT,BCDTAB(400),FOUND,FINAL,UPLIMT,DUMMY(480),R
     /ETURN,
     1BEGIN,BUFENT(40),CONDSN(40,7),ENTRY(150,7),EXENT(40,40),PR
     /EVNO(150
     2 ),RULE,RETFRM,STOP,BLANKS,BUFLST(40),GOTO,STARS,LIST(40,4
     /0),EXTRA
     2(40),NUMB(32),ELSE,STMNT(150),HEADER(100),CONENT(10),ACTEN
     /T(10),
     3ENTRYS(7)
C     EXTRA WILL BE USED IN CASE OF LOOPING
```

```
      COMMON/RULES/NUMB,ELSE,LOGDT,NOSUBT
      DATA RETURN/6HRETURN/
      DATA BLANKS/5H     /
C     FINAL HAS NO SPECIAL MEANI5G HERE.IT IS USED TO ECONOMISE
C     IN SPACE.THIS IS A POINTER USED IN PHASE1 AND THE SAME HAS
C     BEEN USED IN PHASE2
      COMMON/SOURCE/BUFENT,BUFLST,EXTRA,DUMMY,FOUND,INITAL,FINAL
     /,UPLIMT
      EQUIVALENCE (MBRNCH,DUMMY(480)),(CONENT(1),DUMMY(1)),(ACTE
     /NT(1),
     1 DUMMY(11)),(ENTRYS(1),DUMMY(21)),(IMIN1,DUMMY(28)),(NOSTR
     /O,DUMMY
     2(29)),(KRULES,DUMMY(30))
      COMMON/DETAB/BEGIN,GOTO,STOP ,STARS
      COMMON/LOOKUP/BCDTAB
C     ***********************************************************
C
C     THIS HELPS IN GETTING THE ENTRY FOR THE NEXT RULE
C
      SAVERT=RETFRM
      LFRET=LF
C
C     MUST SAVE  AND THEN RESTORE BEFORE RETURN
C     ***********************************************************
      ITER=0
C     FINAL IS 1 IN THE CASE OF A NORMAL CARD, IS 2 IF LOOPING
      IF(RETFRM.EQ.0)  GOTO 3002
      DO 100 LJK=1,LMIN1
      EXTRA(LJK)=EXENT(LJK,RULE)
      BUFENT(LJK)=EXENT(LJK,RULE)
100   EXENT(LJK,RULE)=BLANKS
293   DO  253  J3=1,NCRD
       IRET=RETFRM
      IF(ITER.GT.(NCRD+1))  GOTO 3000
      IF(RETFRM.EQ.BCDTAB(J3))  GO TO 2531
253   CONTINUE
      PAUSE 66660
      STOP
2531  CONTINUE
      ITER=ITER+1
256   J=J3
      LF=LF-1
      IF(ENTRY(J,1).EQ.STOP.OR.ENTRY(J,1).EQ.GOTO)  GO TO 2352
      IF(ENTRY(J,1).EQ.RETURN)  GOTO 2352
      IF(ENTRY(J,1).EQ.LOGDT)  GOTO 2349
      DO  2350  LJK=1,LMIN1
      DO  23501  IJK=1,7
      IF(ENTRY(J,IJK).NE.  CONDSN(LJK,IJK))  GO TO 2350
23501 CONTINUE
```

```
      GO TO 2351
2349  REWIND  4
      MBRNCH=STMNT(J)
      DO 200 I=1,NOSUBT
      IF(MBRNCH.EQ.HEADER(I))  GOTO 300
200   CONTINUE
      PRINT 210,MBRNCH
210   FORMAT(1H ,*MACHINE ERROR IN DTENT,MBRNCH=*,A10)
      STOP
300   IF(I.GT.1) IMIN1=I-1
      IF(I.EQ.1)  GOTO 600
      DO 500 K=1,IMIN1
      READ(4,62985) HEADER(K),NOSTRO,KRULES
62985 FORMAT(A5,2I4)
      DO 400 L=1,NOSTRO
      READ(4,62987) (ENTRYS(J),J=1,5),(CONENT(J),J=1,KRULES)
400   CONTINUE
      READ(4,62989) (ACTENT(J),J=1,KRULES)
500   CONTINUE
62989 FORMAT(*GO TO*,26X,16A6)
62987 FORMAT(5A6,1X,16A6)
C      SEE MERGE FOR DETAILS AND LOGIC
600   READ(4,62985) HEADER(I),NOSTRO,KRULES
      DO 1000KK=1,NOSTRO
      READ(4,62987) (ENTRYS(M),M=1,5),(CONENT(M),M=1,KRULES)
      DO 800 K=1,LMIN1
      DO 760 I=1,5
      IF(ENTRYS(I).NE.CONDSN(K,I))  GOTO 800
760   CONTINUE
      EXENT(K,RULE)=0
      GO TO 1000
800   CONTINUE
      PRINT 810,(ENTRYS(M),M=1,5)
810   FORMAT(1H ,*IN DTENT CAME ACROSS A CONDITION WHICH DOES NO
1000  CONTINUE

      J=J3
      GOTO 2352
2350  CONTINUE
2351  EXENT(LJK,RULE)=0
C     THIS IS JUST TO DISTINGUISH IT FROM DASH ETC
      J=J3
2352  RETFRM=PREVNO(J)
      IF(LF.EQ.0)  LF=LFRET
      IF(RETFRM.EQ.BEGIN)  GO TO 294
      IF(J.EQ.1)  GOTO 294
C     INSERTED ON APRIL 3,1969
      GO TO 293
294   DO  295  LMN=1,LMIN1
```

```
      IF(EXENT(LMN,RULE).EQ.0)  EXENT(LMN,RULE)=BUFENT(LMN)
295   CONTINUE
      FINAL=1
296   LF=LFRET
      RETFRM=SAVERT
3000  FINAL=2
      FINAL=2
      UPLIMT=1
C     THIS WILL BE USED IN FORTAB
      DO 3010 I=1,LMIN1
3010  EXENT(I,RULE)=EXTRA(I)
      GOTO 296
3002  PRINT 3004
3004  FORMAT(1H ,*BECAUSE RETFRM IS 000000 RETURNING TO MAIN*)
      RETURN
      END
$IBFTC PATERN
      SUBROUTINE PATERN(NOOFDL,J1,M1,NAME1,J2,M2,NAME2,J3,M3,
     1NAME3,J4,M4,NAME4,J5,M5,NAME5)
C     STRUCTURE OF ARGUMENTS IS SIMILAR TO PDUMP. UPPER LIMIT OF
C     NOOFDL IS 5.NOOFDL IS TOTAL NO. OF DELIMITERS SPECIFIED IN
C     CALL STATEMENT.
      INTEGER JJ(5),KK(5),NAME(5),DLIMTR(11),BUFFER(600),FINAL,
     1ALNUM(36),FARMAT(3),FOUND,UPLIMT,UPTO
     1,FARMAT(3),FOUND,UPLIMT,UPTO
      COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
      DATA FARMAT(1),FARMAT(3)/2H(A,1H)/
      DATA DLIMTR/1H=,1H,,1H(,1H),1H-,1H ,1H+,1H/,1H.,1H*,2H**/
      DATA ALNUM/1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,1H0,1HA,1HB
     /,1HC,1HD
     1,1HE,1HF,1HG,1HH,1HI,1HJ,1HK,1HL,1HM,1HN,1HO,1HP,1HQ,1HR,
     /1HS,1HT,1
     1HU,1HV,1HW,1HX,1HY,1HZ/
      PRINT 1,(BUFFER(K),K=7,72)
1     FORMAT(1H ,72A1)
      JJ(1)=J1
      JJ(2)=J2
      JJ(3)=J3
      JJ(4)=J4
      JJ(5)=J5
      KK(1)=M1
      KK(2)=M2
      KK(3)=M3
      KK(4)=M4
      KK(5)=M5
      NAME(1)=NAME1
      NAME(2)=NAME2
      NAME(3)=NAME3
      NAME(4)=NAME4
      NAME(5)=NAME5
```

```
      PRINT 2,JJ,KK,NAME
2     FORMAT(1H ,*VALUE OF ARGUMENTS ARE.  JJ...*,5I4,*KK...*,5I4,
     1,*NAME...*,5A6//)
C     THE PROGRAMMER MUST TAKE CARE OF MATCHING ( AND ).
      UPTO=INITAL+UPLIMT-1
      DO  400 J=1,NOOFDL
      KKKK=KK(J)
      JJ3=JJ(J)
      DO  100 I=INITAL,UPTO
      IF(KKKK-26) 10,20,30
10    K=1
      KKK=10
      GOTO170
20    K=11
      KKK=36
      GOTO170
30    K=1
      KKK=36
170   DO 100 JK=K,KKK
      IF(JJ3.NE.11) GOTO180
C     INSTEAD OF 10 WE COULD KEE7 (JJ3-1) AND DO AWAYA WITH FIRST IF
      IF(BUFFER(I).EQ.ALNUM(JK).AND.BUFFER(I+1).EQ.DLIMTR(10 )
     /.AND.
     1BUFFER(I+2).EQ.DLIMTR(10 )) GOTO200
C     THIS IS  THE  CA S E   OF  TESTING  **
180   IF(BUFFER(I).EQ.ALNUM(JK).AND.BUFFER(I+1).EQ.DLIMTR(JJ3))
     / GOTO 200
100   CONTINUE
105   FORMAT(///1H ,*JK=*,I2,2X,*I*,I2,2X,*JJ3=*,I2,2X,*BUFFER(I)
     1=*,A6,* ALNUM(JK)=*,A6//)
      FOUND=0
      PRINT 1000,DLIMTR(JJ3),JJ3
1000  FORMAT(1H ,*SEARCH FAILS  *,A4,2X,*JJ3=*,I2)
      RETURN
200   ITEMP=INITAL
      INITAL=I+2
C     I IS UPTO DELIMITER-1 WHENCE PLUS 2
      IF(JJ3.EQ.11)INITAL=INITAL+1
C     ADDITIONAL FOR EXTRA **
      K1=I-INITAL+1
      FOUND=100
      FINAL=I
      WRITE(99,250)(BUFFER(I),I=ITEMP ,FINAL)
250   FORMAT(6A1)
      CALL BINBCD(K1,FARMAT(2))
      READ(99,FARMAT) NAME(J)
      PRINT 300,INITAL,DLIMTR(JJ3),NAME(J),FINAL
300   FORMAT(//1H ,*SEARCH SUCESSFUL*,*INITAL=*,I5,3X,2H//,A2,2H
     1//,*RESULT OF SEARCH IS   *,A6,3X,* FINAL=*,I5)
```

```
400   CONTINUE
      PRINT 500,INITAL,NAME,FINAL
500   FORMAT(//1H ,*TOTAL SEARCH SUCESSFUL .INITAL=*,I5,5(3X,A6)
     1,3X,* FINAL=*,I5)
      RETURN
      END
```

APPENDIX IV

DECISION TABLES FOR ACTUAL PROGRAMS

One standard question has been, 'Can the Analysis Program analyse itself?' The purpose of this appendix is to present decision tables of some of the subroutines of LOGITRAN, produced automatically by the abalysis program. The listings of the subroutines along with decision tables is given here. Tables A and B have been suppressed.

```
$JOB    CCS036,NAME VGUPTA$ DT'S FOR SUBS OF  LOGITRAN
$IBJOB LGTRAN   NOSOURCE
$RELOAD         U05
      SUBROUTINE  PUTBRN(OPRATR,GOAL,BRNCH1,BRNCH2,BRNCH3,  N)
C ************************************************************************
C
C
C     THIS SUBROUTINE PUTS THE PROPER VALUE IN BRANCHES
C
C
C ************************************************************************
      INTEGER BUFFER(600)
      INTEGER FOUND,FINAL,UPLIMT
      INTEGER ORR,ANDD,NOTT
      INTEGER OPRATR,GOAL,BRNCH1,BRNCH2,BRNCH3,LEZ,LTZ,EQZ,GEZ,G
     /TZ,NEZ,
     1STARS
      DATA  LEZ,LTZ,EQZ,GEZ,NEZ,STARS,GTZ/
     1 4H.LE.,4H.LT.,4H.EQ.,4H.GE.,4H.NE.,5HXXXXX,4H.GT./
      DATA  ORR,ANDD,NOTT/4H.OR.,5H.AND.,5H.NOT./
C
      COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
      IF(OPRATR.EQ.LEZ)  GO TO 512
      IF(OPRATR.EQ.LTZ)  GO TO 510
      IF(OPRATR.EQ.EQZ)  GO TO 520
      IF(OPRATR.EQ.GEZ)  GO TO 523
      IF(OPRATR.EQ.GTZ)  GO TO 530
      IF(OPRATR.EQ.NEZ)  GO TO 513
      IF(OPRATR.EQ.ORR)  GOTO400
      IF(OPRATR.EQ.NOTT)  GO TO 450
      IF(OPRATR.EQ.ANDD)  GO TO 470
      FOUND=4
500   CONTINUE
      RETURN
400   FOUND=2
C     FOUND IS RESPECTIVELY 1 FOR AND,2 FOR OR,3 FOR NOT AND
C     4 FOR ILLEGAL OPCODE
C     FOUND IS 100 FOR RELATIONAL OPS
      GOTO500
450   FOUND=3
      GOTO500
470   FOUND=1
      GOTO500
510   BRNCH1=GOAL
      BRNCH2=STARS
      BRNCH3=STARS
      GO TO 780
```

```
512     BRNCH1=GOAL
        BRNCH2=GOAL
        BRNCH3=STARS
        GO TO 780
513     BRNCH1=GOAL
        BRNCH3=GOAL
        BRNCH2=STARS
        GO TO 780
520     BRNCH2=GOAL
        BRNCH1=STARS
        BRNCH3=STARS
        GO TO 780
523     BRNCH2=GOAL
        BRNCH3=GOAL
        BRNCH1=STARS
        GO TO 780
530     BRNCH3=GOAL
        BRNCH2=STARS
        BRNCH1=STARS
        GO TO 780
780     FOUND=100
        RETURN
        END
DEKEND
```

<u>Subroutine</u> <u>PUTBRN</u>

DECISION TABLE

RULES

TION	1	2	3	4	5	6	7	8	9	10
R-LEZ	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.EQ.0
R-LTZ	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.EQ.0	
R-EQZ	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.EQ.0		
R-GEZ	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.EQ.0			
R-GTZ	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.EQ.0				
R-NEZ	.NE.0	.NE.0	.NE.0	.NE.0	.EQ.0					
R-ORR	.NE.0	.NE.0	.NE.0	.EQ.0						
R-NOTT	.NE.0	.NE.0	.EQ.0							
R-ANDD	.NE.0	.EQ.0								
ONS										
0	RETURN	470	450	400	513	530	523	520	510	512
0		500	500	500	780	780	780	780	780	780
0		RETURN	RETURN	RETURN	RETURN	RETURN	RETURN	RETURN	RETURN	RETURN

Decision Table for Subroutine PUTBRN

```
$JOB    CCS036,NAME VGUPTA$ DT'S FOR SUBS OF  LOGITRAN
$IBJOB LGTRAN   NOSOURCE
$RELOAD         J05
        SUBROUTINE SEQUNC(LIST,M,JPREV)
        INTEGER BUFFER(600),LIST(20),DLIMTR(10)
        INTEGER FOUND,FINAL,UPLIMT
        DATA DLIMTR /1H.,1H(,1H),1H,,1H=,1H*,1H-,1H+,1H/,1H /
        COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
C       FROM EFFICENCY POINT OF VIEW DELIMETERS SHOULD BE ARRANGED
C       IN THE SEQUENCE OF THEIR ORDER OF OCCURANCE
C
C       FOR A GIVEN BUFFER IT WILL GIVE US  THE LIST OF DELIMETERS
C       THAT APPEAR BETWEEN COLS. 7-72 FROM LEFT TO RIGHT.ON ADHOC
C       BASIS THE TOTAL NO. OF DELIMITERS IS TAKEN AS 30 WHICH
C       WILL FORM LIST
        DO 20 I=1,20
20      LIST(I)=0
        KK=INITAL
        M=1
50      DO  100 K=KK,JPREV
        DO 100 J=1,9
        IF(BUFFER(K).EQ.DLIMTR(J))  GO TO 200
        IF(BUFFER(K).EQ.DLIMTR(10))  GOTO 400
        IF(M.GT.20)  GO TO 300
100     CONTINUE
        IF(M.GE.2)  GOTO 400
        RETURN
C
400     M=M-1
        RETURN
200     LIST(M)=DLIMTR(J)
        M=M+1
        KK=K+1
        GO TO 50
300     PRINT 350
350     FORMAT(/1H ,*NO OF DELIMITERS HAS EXCEEDED THE STIPULITED
       1NUMBER OF 20  */)
        RETURN
        END
DEKEND
```

Subroutine SEQUNC

DECISION TABLE

RULES

ENTRY CONDITION	1	2	3	4	5
BUFFER(K)-DLIMTR(J)	.NE.0	.NE.0	.NE.0	.NE.0	.EQ.0
BUFFER(K)-DLIMTR(10)	.NE.0	.NE.0	.NE.0	.EQ.0	
M - 20	.LE.0	.LE.0	.GT.0		
M - 2	.LT.0	.GE.0			
ACTIONS					
GO TO	RETURN	400	300	400	200
GO TO		RETURN	RETURN	RETURN	50
GO TO					SELF

Decision Table for Subroutine SEQUNC.

```
$JOB    CCS036,NAME VGUPTA$ DT'S FOR SUBS OF  LOGITRAN
$IBJOB LGTRAN   NOSOURCE
$RELOAD         U05
       SUBROUTINE   GETENT(ENTRYS,MAXNOP)
       INTEGER  BUFFER(600),ENTRYS(20),FINAL,FARMAT(5),FULWRD,RIG
      /HTP,
      1REMAIN,MAXNOP,UPLIMT,UPTO ,UNPAIR
       INTEGER FOUND
       DATA LEFTP,RIGHTP,FARMAT(1),FARMAT(3),FARMAT(5)/1H(,1H),1H
      /(,4HA6,A
      1,1H)/
       COMMON/SOURCE/BUFFER,FOUND,INITAL,FINAL,UPLIMT
C *****************************************************************
C      THIS SUBROUTINE IS TO BE USED FOR IFS AND COMPUTED GOTOS
C      TO GET THE CHARACTER SET WITHIN THE MATCHING PARANTHISIS
C      THAT IS WELL FORMED EXPRESSIONS.IT ASSUMES THAT LAST
C      CHARACTER ENCOUNTERED IS A ( AND THE NEXT CH. OF BUFFER
C      IS AT INITAL. WHEN RETURNI5G LAST CHARACTER IS AT FINAL
C      ) OF EXPRESSION)
C *    FOUND WILL BE USEFUL FOR CONTINUATION CARDS OF IF
C *****************************************************************
       MAXNOP=1
       UNPAIR=1
       UPTO=INITAL+UPLIMT-1
       DO  100 I=INITAL,UPTO
       IF(BUFFER(I).EQ.LEFTP)  GO TO 150
       IF(BUFFER(I).EQ.RIGHTP)UNPAIR=UNPAIR-1
200    IF(UNPAIR.EQ.0)  GO TO 500
       GO TO 100
150    UNPAIR=UNPAIR+1
       IF(MAXNOP.LT.UNPAIR)  MAXNOP=UNPAIR
       GO TO 200
100    CONTINUE
       FOUND=0
       RETURN
500    K=I-INITAL
C      HERE WE DONOT HAVE TO ADD 1 TO GET PROPER K
       FINAL=I-1
       WRITE(99,600) (BUFFER(I),I=INITAL,FINAL)
600    FORMAT(80A1)
       FULWRD=K/6
       REMAIN=K-FULWRD*6
       CALL  BINBCD(FULWRD,FARMAT(2))
       CALL  BINBCD(REMAIN,FARMAT(4))
CFARMAT  FORMAT(14A6,A4)         EXAMPLE
       IF(REMAIN.GT.0)  FULWRD=FULWRD+1
       READ(99,FARMAT) (ENTRYS(KK),KK=1,FULWRD)
       FOUND=100
       RETURN
       END
DEKEND
```

DECISION TABLE

RULES

DITION	1	2	3	4	5	6	7	8	9	10
FER(I)-LEFTP	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.EQ.0	.EQ.0	.EQ.0	.EQ.0
FER(I)-RIGHTP	.NE.0	.NE.0	.NE.0	.EQ.0	.EQ.0	.EQ.0				
AIR - 0	.NE.0	.EQ.0	.EQ.0	.NE.0	.EQ.0	.EQ.0	.NE.0	.EQ.0	.EQ.0	
AIN - 0		.LE.0	.GT.0		.LE.0	.GT.0		.LE.0	.GT.0	
NOP - UNPAIR							.LT.0	.LT.0	.LT.0	.GE.0
IONS										
TO	100	500	++28	++11	++11	++11	150	150	150	150
TO	RETURN	RETURN	RETURN	100	500	500	++15	++15	++15	SELF
TO				RETURN	RETURN	++28	100	500	500	
TO						RETURN	RETURN	RETURN	++28	
TO									RETURN	

Decision Table for Subroutine GETENT

APPENDIX V

USERS GUIDE

DEKEND is a control card used by LOGITRAN. Characters DEKEND are punched starting from column 1. The remaining portion of the card contains options for suppressing Tables A and B or for Parsing a decision table. This DEKEND should be kept at the end of the program deck, unless one is interested only in Parsing a decision table, in which case this should be the first card in the deck. If columns 7 and beyond of this card are blank, the user's output will consists of Tables A, B, subtables and decision table. In case a user wants to suppress either or both of the tables A and B, he is expected to punch the characters TABLE A and/or TABLE B in the manner shown below:

	1	2
1	0	0
DEKEND	TABLE A	TABLE B

If one is interested in getting a parsed decision table, one has to keep DEKEND card with characters PARSE punched in columns 30 to 34.

	3
1	0
DEKEND	PARSE

This has to be followed by

a card giving the size of decision table fed in (FORMAT 3 I 2), in the order of number of conditions, number of rules, number of action stub's. The actual decision table now follows. Each row of the table is punched as follows:

FORMAT (5A3, 16A4)

CONDITION / ACTION STUB   CONDITION / ACTION ENTRIES

DECK SETUP

LOGITRAN can be called by an:

$EXECUTE   LGTRAN

To have the proper messages for the operator. $X cards have been inserted. The deck setup for getting a decision table for a program is shown in Fig. APP6.

If the message

THE FOLLOWING STATEMENT SEEMS TO BE WRONG IN SYNTAX OR IT HAS NOT BEEN TAKEN CARE OF IN THIS IMPLENETATION.

appears in the output, one should look into the statement. If one is sure that it is a valid FORTRAN statement, then by minor modifications in the program deck, one can get the decision table. An example will make it clear.

Ex. Supposing we had kept the statement as

```
IF (DEPSUM (I). EQ. TOTAL(2) + XYZ) GOTO  13
```

We would have got the above message. We insert the following statements

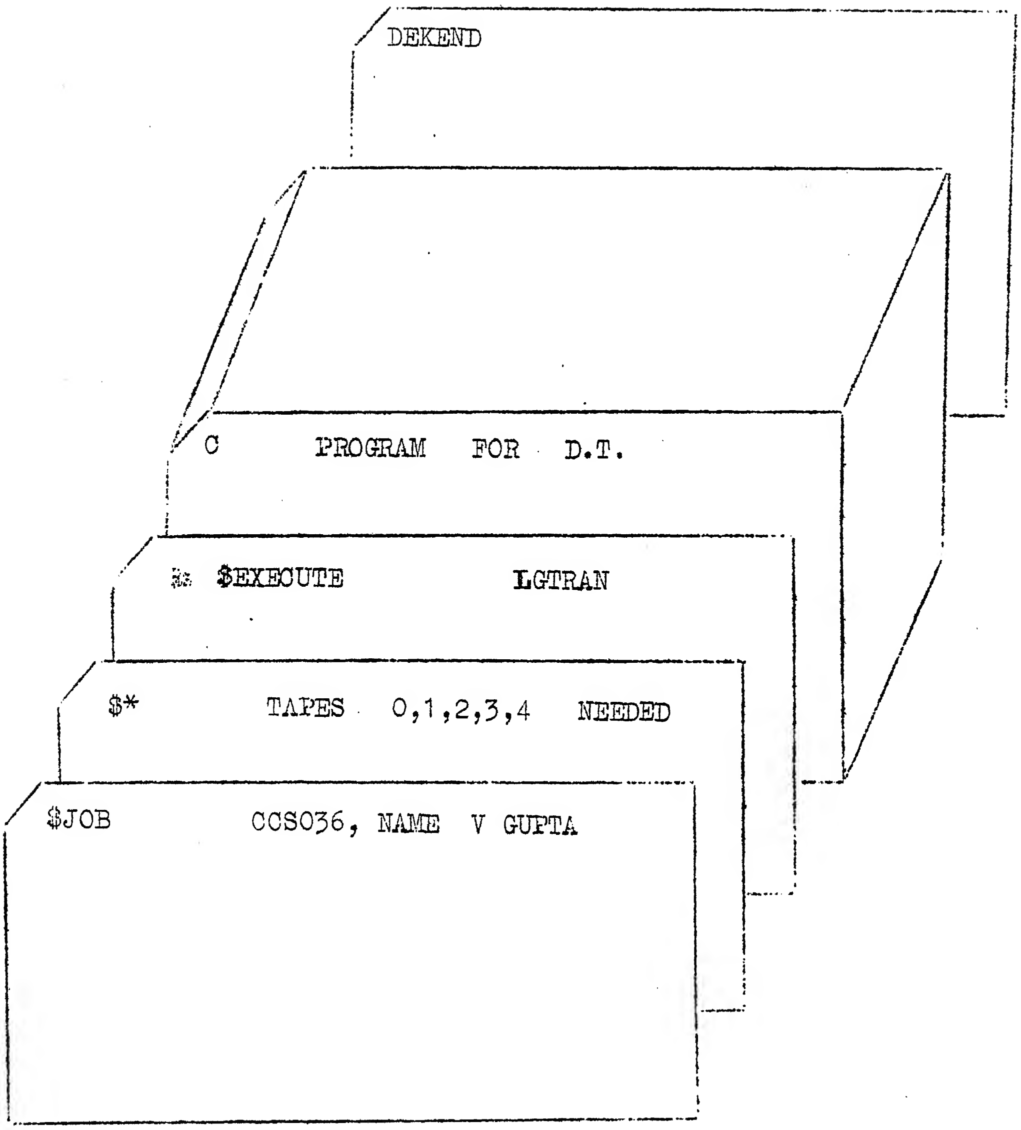


Fig. APP6 DECK SETUP.

instead of the one given above and can get the decision table. The above mentioned message one gets for certain pathological cases.

```
TO2XYZ = TOTAL (2) + XYZ
IF (DEPSUM(I). EQ. TO2XYZ)  GO TO 13
```

Output   In the action set:

(i) a SELF, indicates 'GOTO the same table'

(ii) a ++nnn indicates that the IF statement has an assignment or I/O statement associated with it as an action

(iii) If the action associated is a CALL to a subroutine SUB1, the name of the subroutine appears as an action

Actions indicate the beginning of a series of actions, till the next control statement is reached.

APPENDIX VI

PRACTICAL EXAMPLE

A few decision tables obtained from LOGITRAN have been given in Chapter II, III and Appendix IV. In this appendix is included a practical example, along with its corresponding Decision Table.

The author is very thankful to M/S Union Carbide India Limited, Calcutta for their permission to include this example in this thesis.

```
$JOB    CCS036,TIME008,PAGES030,NAME VGUPTA $UNION CARBIDE
$IBJOB LGTRAN   NOSOURCE
$RELOAD         U05
C
C---------------- ---------- ------------------ -- ------- --
C
C      PRACTICAL PROBLEM FROM  M/S UNION CARBIDE INDIA LIMITED
C      ON LINE BALANCING. INCLUDED WITH THE KIND PERMISSION OF
C      THE MANAGEMENT.
C
C---------------- ---------- ------------------ -- ----------
C
C      PRODUCTION LINE BALANCING - GOPAL+RAO-UNION CARBIDE INDIA LTD
       DIMENSION NM(6)
       DIMENSION IDM(6,10),IMP(6,5),INVMX(6),INV(6),ISPD(6),L(6,5),
      1IMAC(6,5),IDUR(6,5),ISUM(6),ISPDP(6),LINX(6),LINV(6),ITRT(6)
       DO 999 I=1,6
C
       LINV(I)=0
       LINX(I)=0
       DO 1000 J=1,5
       L(I,J)=0
       IMAC(I,J)=0
       IDUR(I,J)=0
 1000  CONTINUE
  999  CONTINUE
       READ1, ((IDM(I,K),K=1,10),I=1,6)
       READ3, (ISPD(I),I=1,6)
       READ3, (INV(I),I=1,6)
       READ3, (INVMX(I),I=1,6)
       READ2, ((IMP(I,J),J=1,5),I=1,6)
       READ 112,(NM(I),I=1,6)
       READ 112,(ITRT(I),I=1,6)
       IT=1
  100  PRINT 8
       WRITE OUTPUT TAPE 3,8
       DO 500 IZ=1,40
       DO 200 I=1,6
       IMNO=NM(I)
       DO 20 J=1,IMNO
       I1=ISPD(I)*IMNO
C      IF(IT-L(I,J)) 20,13,13
       LIJ=L(I,J)
       IF(IT-LIJ) 20,13,13
   13  IDUR(I,J)=0
       IF(I-1) 12,12,10
C   10 IF(INV(I-1)-LINV(I-1))65,65,66
  10   INVI1=INV(I-1)
```

```
      LINVI1=LINV(I-1)
      IF(INVI1-LINVI1) 65,65,66
   65 IMAC(I,J)=0
      GO TO 20
   66 LINV(I-1)=0
C     IF(INV(I-1)-I1)19,19,12
      IF(INVI1-I1)  19,19,12
C  12 IF(INV(I)-LINX(I)) 212,20,212
12    INVI=INV(I)
      LINXI=LINX(I)
      IF(INVI-LINXI)  212,20,212
C 212 IF(INV(I)-INVMX(I)) 11,18,18
212   INVMXI=INVMX(I)
      IF(INVI-INVMXI) 11,18,18
C     IF(IRAND-IMP(I,J)) 14,14,15
   11 READ INPUT TAPE 2,4,IRAND
      IMPIJ=IMP(I,J)
      IF(IRAND-IMPIJ)  14,14,15
   14 IMAC(I,J)=1
      IDUR(I,J)=0
      IF(I-1)  20,20,215
  215 LINX(I-1)=0
      GO TO 20
   15 READ INPUT TAPE 2,4,IRAND
      DO 16 K=1,10
C     IF(IRAND-IDM(I,K)) 17,17,16
      IDMIK=IDM(I,K)
      IF(IRAND-IDMIK)  17,17,16
   16 CONTINUE
   17 IMAC(I,J)=0
      IDUR(I,J)=K
      L(I,J)=IT+K
      GO TO 20
   18 DO 70 K=1,IMNO
   70 IMAC(I,K)=0
      PRINT 6,I,I
      LINX(I)=INV(I)
      WRITE OUTPUT TAPE 3,6,I,I
      GO TO 20
   19 DO 72 K=1,IMNO
   72 IMAC(I,K)=0
      LINV(I-1)=ISPD(I-1)*ITRT(I-1)*NM(I-1)
      M=I-1
      PRINT 7,M,I
      WRITE OUTPUT TAPE 3,7,M,I
   20 CONTINUE
  200 CONTINUE
```

```
      DO 300 I=1,6
  300 ISUM(I)=0
      DO 30 I=1,6
      DO 30 J=1,5
      ISUM(I)=ISUM(I)+IMAC(I,J)
   30 CONTINUE
      DO 40 I=1,6
   40 ISPDP(I)=ISPD(I)*ISUM(I)
      DO 45 I=1,5
   45 INV(I)=INV(I)+(ISPDP(I)-ISPDP(I+1))
      PRINT 5,IT,
     1(IMAC(1,J),IDUR(1,J),J=1,NM(1)),INV(1),
     2(IMAC(2,J),IDUR(1,2),J=1,NM(2)),INV(2),
     3(IMAC(3,J),IDUR(1,3),J=1,NM(3)),INV(3),
     4(IMAC(4,J),IDUR(1,4),J=1,NM(4)),INV(4),
     5(IMAC(5,J),IDUR(1,5),J=1,NM(5)),INV(5)
      WRITE OUTPUT TAPE 3,5,IT,
     1(IMAC(1,J),IDUR(1,J),J=1,NM(1)),INV(1),
     2(IMAC(2,J),IDUR(1,2),J=1,NM(2)),INV(2),
     3(IMAC(3,J),IDUR(1,3),J=1,NM(3)),INV(3),
     4(IMAC(4,J),IDUR(1,4),J=1,NM(4)),INV(4),
     5(IMAC(5,J),IDUR(1,5),J=1,NM(5)),INV(5)
      IT=IT+1
  500 CONTINUE
      WRITE OUTPUT TAPE 3,9
      PRINT 9
C     IF(SENSE SWITCH 3)100,111
C     INSTEAD OF ABOVE CARD INSERTED THE ONE BELOW
      IF(SSW3)  GOTO 100
  111 END FILE 3
C
C     STOP WAS NOT THERE IN THE ORIGINAL PROGRAM.INSERTED
C
      STOP
    1 FORMAT(10I5)
    2 FORMAT(5I5)
    3 FORMAT(6I8)
    4 FORMAT(I5)
    5 FORMAT(I5,
     1213,I7,
     2213,I7,
     3613,I7,
     4213,I7,
     58I3,I7)
    6 FORMAT(5H DIAL,I2,28H IS FULL HENCE STOP MACHINE   ,I2//)
    7 FORMAT(5H DIAL,I2,29H IS EMPTY HENCE STOP MACHINE    ,I2//)
    8 FORMAT(///)
    9 FORMAT(1H1)
  112 FORMAT(6I2)
      END
```

DECISION TABLE

RULE

ENTRY CONDITION	1	2	3	4	5	6	7	8	9	10
IT -LIJ	.LT.0	.LT.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0
SSW3	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	.EQ.0
I -1			.LE.0	.LE.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0
INVI -LINXI			.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0
INVI -INVMXI			.LT.0	.LT.0	.LT.0	.LT.0	.LT.0	.LT.0	.LT.0	.LT.0
IRAND-IMPIJ			.LE.0	.LE.0	.LE.0	.LE.0	.GT.0	.GT.0	.GT.0	.GT.0
IRAND-IDMIK							.LE.0	.LE.0	.GT.0	.GT.0
INVI1-LINVI1										
INVI1-I1										
ACTIONS										
GO TO	20	100	13	100	215	100	15	100	16	100
GO TO	STOP	SELF	12	SELF	20	SELF	17	SELF	20	SELF
GO TO			212		STOP		20		STOP	
GO TO			11				STOP			
GO TO			14							
GO TO			20							
GO TO			STOP							

(Continued on next page)

CONDITION	RULE 11	12	13	14	15	16	17	18	19	20
IT -LIJ	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0
SSW3	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	.EQ.0
I -1	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.LE.0	.LE.0
INVI -LINXI	.NE.0	.NE.0	.EQ.0	.EQ.0					.NE.0	.NE.0
INVI -INVMXI	.GE.0	.GE.0							.LT.0	.LT.0
IRAND-IMPIJ									.LE.0	.LE.0
IRAND-IDMIK										
INVI1-LINVI1					.LE.0	.LE.0	.GT.0	.GT.0	.GT.0	.GT.0
INVI1-I1							.LE.0	.LE.0	.GT.0	.GT.0
ACTIONS										
GO TO	18	100	20	100	10	100	66	100	12	100
GO TO	20	SELF	STOP	SELF	65	SELF	19	SELF	212	SELF
GO TO	STOP				20		STOP		11	
GO TO					STOP				14	
GO TO									20	
GO TO									STOP	
GO TO										

(Continued on next page)

CONDITION	RULE 21	22	23	24	25	26	27	28	29	30
IT -LIJ	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0	.GE.0
SSW3	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	.EQ.0	.NE.0	.EQ.0
I -1	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0
INVI -LINXI	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.NE.0	.EQ.0	.EQ.0
INVI -INVMXI	.LT.0	.LT.0	.LT.0	.LT.0	.LT.0	.LT.0	.GE.0	.GE.0		
IRAND-IMPIJ	.LE.0	.LE.0	.GT.0	.GT.0	.GT.0	.GT.0				
IRAND-IDMIK			.LE.0	.LE.0	.GT.0	.GT.0				
INVI1-LINVI1	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0
INVI1-I1	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0	.GT.0
ACTIONS										
GO TO	215	100	15	100	16	100	18	100	20	100
GO TO	20	SELF	17	SELF	20	SELF	20	SELF	STOP	SELF
GO TO	STOP		20		STOP		STOP			
GO TO			STOP							
GO TO										
GO TO										
GO TO										